Universidad Carlos III de Madrid

Departamento de ingenería informática

Proyecto fin de carrera:

Estudio del cambio de representación en planificación automática

Autor: Adrián Gil Moral

Tutora:
Raquel Fuentetaja Pizán

Madrid, junio de 2017

Título: Estudio del cambio de representación en planificación automática
Autor: Adrián Gil Moral
Tutora: Raquel Fuentetaja Pizán

EL TRIBUNAL

Presidenta: María Belén Ruiz Mezcua

Vocal: Luis Enrique Moreno Lorente

Secretario: Andrea Bellucci

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día 5 de julio de 2017 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTA

Agradecimientos

A mi familia, por proveerme de *una habitación propia*.

A Rafa, mi *StackOverflow* particular.

A Raquel, por toda su paciencia y dedicación para con este proyecto.

A todas las que madrugan para defender el derecho a *una habitación propia*.

Hay que ser un fuera de serie para venir de un entorno de clase trabajadora como Daniel [uno de mis compañeros de clase] y aspirar a algo diferente. "No sé si alguna vez conseguí llegar a ese tipo de gente". Siente que por más que lo intentaran, muchos de los padres de los padres de los niños de la clase trabajadora con problemas de aprendizaje difícilmente podían ayudarles. Los propios padres a menudo tenían problemas de aprendizaje, así que no eran capaces de ayudar a sus hijos. La gente de entornos cultos de clase media tienen padres que pueden ayudarles y animarles en sus deberes.

Helena fue, de hecho, una profesora extremadamente buena e inspiradora, pese a las dificultades a las que se enfrentaba en lo que describía como un "colegio duro" y falto de recursos. Pero al final yo fui el único niño que completó la enseñanza secundaria, por no hablar de la universidad. ¿Por qué? Porque nací en una familia de clase media (mi madre era profesora en la Universidad de Salford y mi padre responsable de regeneración económica en el Ayuntamiento de Sheffield). Crecí en un medio culto y me limitaba a seguir los pasos profesionales bien pagados cuando fui a la universidad. No sufrí la inestabilidad y las tensiones que tener que arreglárselas en la vida puede causar a una familia. Vivía en una buena casa. Estas cosas que se le niegan a muchísima gente de clase trabajadora.

OWEN JONES, *Chavs. La demonización de la clave obrera* [2011]

Los Doctores Americanos nos advierten que deben su ciencia a los indios y a los negros: porque si los Señores Doctores hubieran tenido que arar, sembrar, recoger, cargar y confeccionar lo que han comido, vestido y jugado durante su vida inútil... no sabrían tanto: ... estarían en los campos y serían tan brutos como sus esclavos.

SIMÓN RODRÍGUEZ, *Nota sobre el proyecto de educación popular* [1830]

Índice

Índice de figuras

Índice de tablas

1 Resumen

Este es un trabajo de investigación en el que intenta dar luz sobre la posibilidad de reducir el tiempo de búsqueda y mejorar la calidad de las soluciones mediante variaciones en la representación de los dominios y problemas de planificación automática. Para ello, se ha realizado un estudio empírico donde se han utilizado tanto versiones modificadas a mano de los dominios y problemas como versiones generadas mediante software independiente de dominio desarrollado por otros investigadores.

Con este trabajo se pretende demostrar lo ligado que está la planificación a la representación de los dominios, así como la enorme influencia que ejerce el planificador utilizado en la representación adecuada. Es decir, la idoneidad de determinadas representaciones de dominios puede variar según el planificador utilizado.

La conclusiones que se obtienen de este trabajo es que se confirma la hipótesis inicial de que la variación de la representación de los dominios puede hacer que varíen significativamente los tiempos de búsqueda y la calidad de la solución encontrada; que estos resultados son dependientes del tipo de planificador utilizado y que aún queda un largo trecho por recorrer en el terreno de la investigación cuanto al desarollo software de modificación automática de dominios independiente de dominio puesto que los resultados obtenidos mediante el uso de los mismos han sido sobrepasados holgadamente (en términos de calidad de la solución y menor tiempo de búsqueda) por varias de las versiones generadas a mano.

Palabras clave: **planificación automática**, **representación en planificación automática**, **PDDL**, **Metric-FF**, **LPG-td**

2 Glosario

Tabla 1: Glosario de acrónimos

Acrónimo	Significado
ADL	Action Description Language
A-graph	Action graph
AP	Automated Planning
EHC	Enforced Hill Climbing
FF	Fast-Forward
GB	GigaBytes
GNU	Gnu Not Unix
GPL	General Public License
GPS	General Problem Solver
IBM	International Business Machines
ICAPS	International Conference on Automated Planning and Scheduling
IPC	International Planning Competition
LPG	Local Search Planner
LPG-td	Local Search Planner Timed initial literals and Derived predicate
MFF	Metric Fast-Forward
PDDL	Planning Domain Definition Language
PDF	Portable Document Format
PTT	Planning Task Transformer
STRIPS	Stanford Research Institute Problem Solver
TA-graph	Temporal Action graph

3 Introducción

Este es un trabajo de investigación en el que se intenta analizar la correlación que existe en planificación automática entre distintas formas de representar el mismo problema a resolver y el resultado obtenido con la planificación de estas distintas versiones con varios planificadores.

La planificación automática consiste en la resolución de problemas de búsqueda con un resolutor de tareas genérico que en caso de resolver el problema devuelva una serie de pasos que lleven de un estado inicial al estado final o meta, siendo ambos estados definidos como entrada para los planficadores. Este área de investigación pertenece a la inteligencia articial y está potenciada principalmente por conferencias y competiciones como ICAPS e IPC. Siendo de este último evento de donde se han sacado los planificadores y el conjunto de representaciones de problemas utilizadas en este trabajo.

El estado del arte de la investigación que se ha hecho en planificación automática está centrada principalmente en la mejora de algoritmos y heurísticas de planificación, siendo exiguos los trabajos que tienen como enfoque el cambio de representación como forma para mejorar la planificación automática. Es en el cambio de representación de dominios y problemas de planificación automática como medio para mejorar los resultados de dicha planificación en lo que se ha centrado este trabajo.

En las próximas secciones se encontrará un desarrollo del estado del arte de este área de investigación, una descripción de todas las representaciones que se van a utilizar en este trabajo, un testeo de las mismas con diversos planificadores para intentar ver qué mejoras se obtienen con cada representación y finalmente se extraerán conclusiones relativas a los resultados que se ha obtenido.

Por último, en los anexos se podrá encontrar información adicional a este trabajo, a saber: la definición exacta de todos los dominios y problemas utilizados, la planificación temporal y económica de este proyecto, sus regulaciones y un resumen en inglés de todo este trabajo.

4 Objetivos

Antes de establecer los objetivos de este trabajo, se parte primero de la base de que pueden existir múltiples representaciones de dominios y problemas de entrada para un mismo tipo de problema de planificación automática. Esta hipótesis inicial queda ampliamente demostrada en la bibliografía referente a muchos de los modificadores independientes de dominio como puede ser *PTT* o *Baggy*, que han sido ambos utilizados en las pruebas de este proyecto y cuyo funcionamiento queda descrito en la sección del estado del arte. Adicionalmente, esto queda demostrado en el momento que se realiza una modificación de la representación de un problema sin que esto cambie la lógica de dicho problema.

Partiendo de esta premisa, el objetivo principal de este trabajo está enmarcado en el estudio del impacto que tienen estos cambios de representación en la planificación automática para así poder obtener soluciones a problemas de planificación en menos tiempo de búsqueda con igual o mejor calidad de las soluciones. Es decir, conociendo el tipo de modificaciones que son favorables para la planificación, se pueden establecer unas pautas para la codificación de representaciones de problemas para la planificación automática o incluso se pueden desarrollar modificadores de representaciones de problemas y dominios de planificación automática independientes de dominio que verifiquen que la representación de los dominios y problemas es la más adecuada y de no ser positiva la respuesta, modifique automáticamente el dominio y/o el problema de entrada para con el fin de que éste o éstos tengan la representación más adecuada en términos de menor tiempo de búsqueda y de calidad de la solución.

Estos dos últimos objetivos sin embargo sobrepasan el trabajo realizado en este proyecto, que como se ha comentado anteriormente, es a día de hoy una cuestión abierta en la investigación de la planificación automática. El objetivo de este proyecto en relación con otros proyectos es por tanto que este proyecto pueda servir a futuros investigadores para iniciar la ruta hacia lo que se considera el fin último en el que estaría englobado este trabajo: conseguir un transformador independiente de dominio que devuelva como salida la mejor representación posible.

Este objetivo se puede definir más concretamente no sólo como la búsqueda de la resolución de la respuesta de si existe correlación entre la representación de un problema y su resolución (pues esto sería sólo la verificación de aquello que ya ha sido constatado por otros investigadores), sino concretamente qué tipo de representaciones pueden ser más o menos beneficiosas para la planificación automática.

Este estudio se realizará con el lenguaje de representación de planificación automática utilizado la *IPC*, *PDDL*, utilizando el dominio y los problemas de *Childsnack* presentados en la *IPC* del 2014. Los subobjetivos de este trabajo para la consecución del objetivo principal descrito en el párrafo anterior son los siguientes:

1. Estudio del estado del arte de algunos de los últimos modificadores automáticos de representación en *PDDL*.
2. Realización de modificaciones con modificadores automáticos de terceros del dominio *Childsnack*.
3. Realización de modificaciones manuales del dominio *Childsnack*.
4. Ejecución de la planificación de tanto los dominios generados automáticamente como los realizados a mano con los planificadores *Metric-FF*, *LPG-td* y *Fast-Downward* con un límite de tiempo de 30 minutos por cada problema, tal y como establecen las normas de las competiciones de la *IPC*.
5. Estudio de la correlación entre las distintas versiones y los distintos planificadores automáticos. Tras esto y dependiendo del resultado obtenido, estudio de las representaciones más adecuadas independientes o dependientes del planificador utilizado.

5 Estado del arte

5.1 Inteligencia Artificial

5.1.1 Definición

Comunmente se consideran las conferencias de Dartmounth de 1956[1] como el gérmen de la inteligencia artificial como campo de investigación.

Estas conferencias tenían como objetivo el estudio durante dos meses por parte de diez académicos de todos los aspectos de la inteligencia para poder describirla con una precisión tal que un ordenador pudiera simular la misma[2]. El organizador de estas conferencias era J. McCarthy, quien acuñaría en estas conferencias el término de inteligencia artificial como *"la ciencia y la ingeniería dedicadas a la creación de máquinas inteligentes, especialmente programas de ordenador inteligentes"*[3].

Cuando el término se popularizó, aparecieron muchas otras definiciones, que Stuart J. Russel y Peter Norvig clasificaron en cuatro grupos[4]:

- Las que definían la inteligencia artificial como sistemas que **piensan como humanos**.
- Las que definían la inteligencia artificial como sistemas que **actúan como humanos**.
- Las que definían la inteligencia artificial como sistemas que **piensan racionalmente**.
- Las que definían la inteligencia artificial como sistemas que **actúan racionalmente**.

5.1.2 ¿Puede una máquina ser inteligente?

A medida que va avanzando el desarrollo tecnológico, va creciendo el debate en torno a si una máquina puede ser inteligente. Entre los distintos criterios para establecer la inteligencia de una máquina, destacan estas tres aportaciones:

- El juego de imitación, más tarde conocido como **el test de Turing**, fue propuesto por Alan Turing en 1950[5]. Consistía en que un evaluador interrogara a través de un teletipo a una máquina y a una persona que tuviesen cada uno un canal distinto e identificable de comunicación con el evaluador. Si el evaluador no fuese capaz de discernir cuál de los dos es la máquina el 70% del tiempo durante cinco minutos, entonces se consideraría que la máquina es inteligente.

Siguiendo por esta línea, desde 1991 tiene lugar anualmente el premio Loebner, [6] en el cual distintos programas de ordenador compiten entre sí con el fin de intentar engañar durante el máximo tiempo posible al evaluador del test de Turing. Anualmente se entrega la medalla de bronce al programa más parecido a un ser humano. La medalla de plata (para el programa que sea capaz de superar el test de Turing) y la medalla de oro (para el programa que pueda superar el test de Turing añadiendo las entradas visuales y auditivas) aún no han sido entregadas en ninguna de las ediciones de la competición.

- **La objeción de Lady Lovelace**, término acuñado por Alan Turing en el ya mencionado artículo *Computing machinery and intelligence*, cuestiona que las máquinas sean capaces de crear o aprender de manera autónoma, puesto que, en palabras de Lady Lovelace *"[La máquina] puede hacer lo que sea que sepamos ordenarle"*, sin embargo *"nunca puede hacer algo realmente nuevo"*[7].

- **La paradoja de la caja china de Searle**, también conocido como la habitación china, es un experimento mental propuesto por John Searle y popularizado por Roger Penrose que rebate la validez del test de Turing a la vez que plantea la incapacidad de pensar de las máquinas. El experimento es el siguiente: supongamos que tenemos una máquina que recibe como entrada un texto en chino, del cual extraiga como salida un segundo texto en este mismo idioma gracias a algún tipo de detección de sintaxis. Supongamos, a su vez, que en una segunda habitación se encuentra un humano que no es capaz de comunicarse en chino, pero que tiene asu disposición una serie de manuales sobre el idioma que le indicarían qué carácteres debería escribir en función de los carácteres de entrada que haya. Pues bien, de cualquiera de estas dos maneras, concluye Searle, podrían hacer creer al evaluador que entienden el idioma de entrada, aunque no sea el caso. De aquí, Searle saca dos posibles conclusiones: o bien la habitación comprende el idioma, o bien el test de Turing no es suficiente como test de inteligencia.

5.1.3 Evolución histórica

Antes de empezar con el desarrollo histórico de la inteligencia artificial, es importante recalcar que esta sección se va a circunscribir exclusivamente a los avances tecnológicos que se dieron en esta materia, obviando por tanto todas las aportaciones mitológicas[8], filosóficas [9]o literarias[10][11].

Aunque la máquina de Turing[12] fue un importante precedente para el desarrollo de la inteligencia artificial, se considera al modelo de neuronas artificiales de 1943

de la mano de Warren McCulloch y Walter Pitts[13] como el primer trabajo en este campo. En él, demostraron que la máquina de Turing podía ser implementado mediante una red finita de neuronas formales.

Más adelante, tuvo lugar en 1956 la ya mencionada conferencia de Dartmounth, a partir de la cual la inteligencia artificial se convirtió en un campo de investigación. Desde esta fecha hasta 1974, se fueron generando altas expectativas respecto a las posibilidades de la inteligencia artificial, lo que provocó la llegada de grandes flujos de financiación a este área de conocimiento.

Dentro de esta época, cabe a su vez destacar los siguientes **avances tecnológicos**:

1957 Aparece el programa General Problem Solver (GPS)[14], diseñado para ser un programa universal de la resolución de problemas. Es el principal antecedente de la planificación automática.

1958 Aparece el lenguaje de programación de alto nivel *LISP*[15], que sería más tarde utilizado para la programación de los primeros sistemas expertos: Mycin[16] y Dendral[17].

1959 **Aparición del perceptrón simple**[18].

1966 Aparición del programa ELIZA[19], que fue uno de los primeros lenguajes en procesar lenguaje natural.

1972 Aparición del lenguaje de programación lógica Prolog[20], que más adelante sería propuesto como el lenguaje nativo de las máquinas de quinta generación[21].

A pesar de todos los avances tecnológicos, no se cumplen las expectativas depositadas en esta rama de conocimiento, por lo que entre 1974 y 1980 se reduce considerablemente la inversión en este nuevo área de conocimiento. Algunos de los problemas con los que se encontraron los científicos fueron la explosión combinatoria (una gran cantidad de problemas únicamente pueden resolverse mediante tiempo exponencial), la limitada capacidad de procesamiento de los ordenadores y la paradoja de Moravec, que venía a decir que *"es fácil comparativamente conseguir que las computadoras muestren capacidades similares a las de un humano adulto en tests de inteligencia, y difícil o imposible lograr que posean las habilidades perceptivas y motrices de un bebé de un año"*[22].

Durante estos años, los avances tecnológicos en este área fueron exiguos. Sin embargo, en el año 1980 la inteligencia artificial volvió a ponerse de moda con

el **proyecto quinta generación**, con la cual el gobierno japonés, tras su éxito en el mercado de la microelectrónica de consumo y en el mercado automovilístico, pretendía liderar la siguiente revolución de computadoras, financiando para ello el proyecto con 850 millones de dólares.

Entre los objetivos perseguidos en las computadoras estaba inventar máquinas con capacidad de conversar, traducir, interpretar imágenes o incluso de razonar.

Sin embargo, al igual que sucedió en el periodo de 1956 a 1974, las expectativas eran demasiado altas, por lo que al no cumplirse lo esperado, a partir de 1987 cayó la financiación hasta la finalización del proyecto en 1993. Posterior a esta época caben destacar los siguientes hitos de la inteligencia artificial:

1997 Una supercomputadora fabricada por IBM, ***Deep Blue***, gana un encuentro a seis partidas contra el actual campeón del mundo de ajedrez por 3 a 2, siendo la primera supercomputadora en lograr este hito[23].

2011 El sistema informático ***Watson*** desarrollado por IBM fue capaz de vencer por primera vez a dos de los mejores jugadores de *Jeopardy!* de la televisión[24].

2015 El sistema informático *AlphaGo* desarrollado por Google DeepMind fue capaz de vencer por primera vez a un jugador profesional en una partida de Go sin utilizar piezas de handicap[25].

5.2 Planificación automática

5.2.1 Definición

De manera sucinta, podríamos definir la planificación automática como *"el arte y la práctica de pensar antes de actuar"*[26]. Si quisiéramos una definición algo más técnica, podríamos a su vez definir la planificación automática como la disciplina de la inteligencia artificial enfocada al desarrollo de solucionadores para un modelos matemáticos bien definidos, siendo estos solucionadores programas que cogen la descripción de una instancia particular de un modelo y automáticamente computan su solución[27].

5.2.2 Elementos comunes de los planificadores

Todos los planificadores en estas tres **características**[28]:

- **El modelo conceptual**, que es la definición formal de la tarea que se va a resolver así como la estructura de la solución.

- **El lenguaje de representación**, que es la manera de definir tanto el problema como el entorno.
- **El algoritmo**, que es la técnica utilizada para resolver el problema.

5.2.3 Planificación clásica

La planificación clásica cuenta con el modelo más básico de planificación automática. En términos matemáticos, el **modelo conceptual de la planificación clásica** se define como la siguiente tupla:

$\Sigma = (S, A, \Upsilon, C)$

- S: conjunto de estados posibles.
- A: conjunto de acciones posibles.
- Υ: es la función de transición entre estados $\Upsilon(s, a, s')$, donde $s' \in S$ es el estado generado tras ejecutar la acción $a \in A$ desde el estado $s \in S$.
- C: es la función de coste $C(a) \forall a \in A$.

Siguiendo con esta definición del modelo conceptual, es posible a su vez definir el **problema de planificación clásica** con la siguiente tupla:

$\prod = (\Sigma, s_0, G)$

- Σ: modelo conceptual.
- s_0: definición del estado inicial donde $s_0 \in S$.
- G: conjunto de estados meta, donde $G \subseteq S$.

La solución para los problemas de planificación clásica será una secuencia ordenada de acciones $M = (a_0, ..., a_n) \forall a_i \in A$ que permitirá realizar una serie de transiciones $(t_0, ..., t_n) \forall t \in \Upsilon$ para así poder llegar del estado inicial s_0 a uno de los estados meta $g \in G$.

Generalmente, la planificación clásica se refiere a la planificación en sistemas restringidos de transición de estados[29], también denominados planificación *STRIPS*. Las características de dicho modelo conceptual son las siguientes:

- **Conjunto de estados finitos**.
- **Entorno completamente observable**. El planificador tiene en todo momento una descripción completa del entorno.

- **Acciones deterministas**, el efecto de cualquier acción es siempre el mismo y se conoce con antelación.
- **Entorno estático**, el entorno únicamente cambia cuando se realiza una acción.
- **Tiempo implícito**, las acciones y los efectos no tienen duración y las transiciones son instantáneas.
- **Planificación fuera de línea**, el planificador no toma en cuenta ningún tipo de cambio que pueda ocurrir en Σ en paralelo a la planificación.

Un problema de planificación clásica puede ser resuelto mediante la transformación de este en otro tipo de problema (como puede ser transformándolo en un problema de tipo SAT[30]) o buscando una solución en el espacio de estados del problema de planificación, siendo esta última técnica la de más extendido uso. Esta técnica está compuesta de estos tres elementos:

- **Grafo de búsqueda**, donde cada nodo representa un estado y cada arista una transición entre estados.
- **Dirección de la búsqueda**, que puede ser hacia adelante o **progresiva** (la búsqueda se realiza del estado inicial al estado o estados meta), hacia atrás o **regresiva** (la búsqueda se realiza desde el estado o estados meta al estado inicial) o **bidireccional** (la búsqueda se realiza paralelamente desde el estado inicial y desde el estado meta). La mayor parte de la investigación se ha centrado en la búsqueda progresiva, puesto que en la búsqueda regresiva se producen muchos más estados espurios (es decir, estados que no se pueden alcanzar desde el estado inicial) que en la búsqueda progresiva y los resultados empíricos obtenidos con la ejecución de ambos métodos son mejores con la búsqueda progresiva.
- **El algoritmo de búsqueda**, que define la forma de búsqueda de la solución en el gráfo de búsqueda.

5.2.4 Planificación heurística

Se entiende planificación heurística como la resolución de problemas de planificación mediante un algoritmo de búsqueda guiado por una función heurística independiente del dominio[31]. La planificación heurística es es el tipo de planificación mayoritariamente usada[32]. La diferencia existente entre la búsqueda de una solución óptima y una satisfacible es que mientras que en el segundo caso únicamente se busca una posible solución, en el caso de la búsqueda de la solución óptima computan de manera distintas las soluciones, siendo necesarias métricas para determinar

el o los parámetros según el cual una solución es más o menos óptima, como puede ser el número de pasos necesarios para alcanzar la solución o el coste de dicha solución. Esto por lo general requiere un mayor coste de recursos, pues la búsqueda no tiene por qué pararse al encontrar la primera solución al problema dado.

Actualmente, las heurísticas utilizadas para la planificación pueden clasificarse en cinco grupos[33]:

- **Relajación de la eliminación**. Se estima el coste a la meta sin tener en cuenta los efectos negativos de las acciones. Algunos ejemplos de estas heurísticas son la heurística admisible *max heuristic* y la heurística no admisible *FF*[34].

- **Abstracción**. Se estima el coste a la meta mediante la resolución de una proyección simplificada del espacio de estados. La función de abstración es la que se encarga de determinar qué estados deben de ser agrupados así como de que no se relaje el problema más de lo necesario (ya que convertiría a la heurística en inservible). A diferencia de las heurísticas de relajación de la eliminación, las heurísticas de abstración son siempre admisibles. Algunos ejemplos de este tipo de heurísticas son las abstracciones cartesianas o los patrones estructurales.

- **Caminos críticos**: estima el coste mediante la longitud de la solución de una versión relajada del problema.

- **Puntos de referencia (*landmarks*)**. son hechos que tienen que ser ciertos en algún punto en todos los planes. El punto de referencia es, por tanto, un conjunto de acciones donde todos los planes contienen al menos una acción de a. La mayoría de heurísticas basadas en puntos de referencia son estimaciones basadas en heurísticas de eliminación. No todas sus heurísticas son admisibles. Algunos ejemplos de heurísticas basadas en puntos de referencia que sí son admisibles son la heurística de *LAMA*, heurísticas basadas en puntos de referencia de coste segmentado (*cost-partitioned landmarks*) y heurísticas basadas en puntos de referencia de corte (*landmark-cut heuristic*).

- **Flujos de red (*network flows*)**. Teniendo en cuenta que el número de veces que un hecho es producido y es consumido debe estar balanceado, es posible realizar un programa lineal para codificar esta información con la que se obtienen los valores de la heurística. Algunos ejemplos admisibles de esta heurística son la heurística de flujo (*flow heuristic*) y la heurística de la ecuación de estado (*state equation heuristic*).

Estas heurísticas pueden después ser combinadas. A la hora de agregar las heurísticas, hay que tener cuidado de no perder las admisibilidad. El método más simple para realizar esto es quedándose en cada estado únicamente con el valor máximo de las distintas heurísticas.

5.2.5 Lenguajes de programación para la planificación

Aunque existen más lenguajes de representación para la planificación que los que se van a nombrar, siendo *PDDL* el lenguaje de más extendido uso así como el lenguaje utilizado para las competiciones de la *International Planning Competition* (*IPC*), únicamente se va hablar de sus principales predecesores (*STRIPS* y *ADL*) y de *PDDL*.

5.2.5.1 STRIPS

Uno de los primeros lenguajes que se utilizó para la planificación clásica fue *STRIPS* (*Stanford Research Institute Problem Solver*)[35]. Al igual que *PDDL*, *STRIPS* utiliza lógica de predicados, también llamada lógica de primer orden. Utilizando este lenguaje, es posible definir el estado inicial, los estados meta y las acciones para poder alcanzar alguno de esos estados. Para ello, se utilizan instancias en forma de tuplas $\langle P, O, I, G \rangle$.

- **P**: es el conjunto de proposiciones.
- **O**: es el conjunto de acciones (también llamadas operadores), cada una de ellas compuestas por una tupla $\langle \alpha, \beta, \gamma, \delta \rangle$:
  - α: condiciones que deben ser verdaderas para la ejecución.
  - β: condiciones que deben ser falsas para la ejecución.
  - γ: condiciones que se hacen verdaderas tras la ejecución.
  - δ: condiciones que se hacen falsas tras la ejecución.
- **I**: es el estado inicial, definido como una serie de proposiciones.
- **G** es el estado meta, definido por la tupla $\langle N, M \rangle$
  - N: representa las condiciones que tienen que ser verdaderas en el estado meta.
  - M: representa las condiciones que tienen que ser falsas en el estado meta.

5.2.5.2 ADL

El lenguaje *ADL* (*Action Description Language*)[36] es posterior a *STRIPS*, y añadía a este último mucha más versatilidad. Algunas de las principales mejoras que incorpora *ADL* respecto a *STRIPS* se citan a continuación:

- El lenguaje *STRIPS* únicamente admite la declaración de literales como positivos, mientras que *ADL* admite la **declaración de literales tanto positivos como negativos**.
- El lenguaje *STRIPS* opera bajo la asunción del mundo cerrado[37], por lo que todos los literales desconocidos son tomados como falsos. Por contra, *ADL* **opera bajo la asunción del mundo abierto**, por lo que el valor de todos los literales desconocidos es tomado como desconocido.
- En *STRIPS*, los objetivos sólo pueden contener conjunciones. En *ADL*, **los objetivos pueden contener conjunciones y disjunciones**.
- En *STRIPS*, no se permiten definir tipos de variables, mientras que en *ADL* sí.
- En *ADL* se permite la introducción de predicados de igualdad.
- En *ADL* **se permiten efectos condicionales**.
- En *ADL* **se permiten literales cuantitativos**.

5.2.5.3 PDDL

El lenguaje *PDDL*[38] estaba originalmente basado en *STRIPS* y *ADL* y surge como intento estandarizarización de los lenguajes de planificación. La versión 1.2 de *PDDL* es utilizada como **lenguaje oficial en la** primera ***IPC*** (*International Planning Competition*) y así ha permanecido en siguientes ediciones de esta competición (cambiando únicamente la versión del lenguaje utilizada).

Los problemas en *PDDL* se dividen en dos partes:

- **Definición del dominio**: es la descipción tanto del entorno como de las acciones. Las acciones se expresan mediante un nombre que la identifique, una serie de precondiciones e información añadida y suprimida tras la ejecución de la acción.
- **Definición del problema**: es la definición de un estado inicial y de un estado meta.

Esta separación también se produce en los ficheros que el programador debe introducir como parámetros, lo que facilita la clasificación de problemas (ya que para este fin únicamente se toma en cuenta la definición del dominio).

Por último, es importante destacar que aunque *PDDL* describe los problemas en lógica de primer órden, no es necesario que los planificadores se ejecuten siguiendo la lógica de primer órden.

5.2.6 Complejidad computacional de la planificación

En esta sección se va a hablar exclusivamente de la complejidad computacional genérica de la planificación; independientemente de los dominios, problemas y sus representaciones respectivas y los algoritmos utilizados para hallar la secuencia de operadores que permite hallar una solución al problema. **La complejidad computacional mínima de cualquier problema de planificación** sería que existiese un único operador con una interfaz de tamaño uno y que tras la aplicación de este operador con un único predicado definido en el problema, se resolviese el problema. Este caso, el de menor complejidad posible de la planificación automática, tendría complejidad $\mathbf{O(1)}$.

A su vez, también podríamos hablar de la **complejidad del caso promedio**. Sin embargo, en este caso al no estar acotado el conjunto de problemas, dominios y sus posibles representaciones, este tipo de complejidad **no puede ser hallada**.

Es por ello que el dato que se considera que la complejidad que más puede ayudar a entender la dificultad de resolver este tipo de problemas es la complejidad máxima de la planificación automática. Esta complejidad computacional, al tratarse de un problema vagamente definido, ya que no se cuenta con una representación de un dominio y un problema que se asegure que vaya a ser el problema más complejo computacionalmente hablando que cualquier otro problema que se pueda proponer, por lo que no es posible acotar de manera precisa esta máxima complejidad, va a ser acotada mediante los tipos de complejidad computacional $P - NP$ en lugar de indicadores de complejidad computacional más exactos como $O(n)$.

Los **tipos de complejidad computacional P-NP** en relación al tiempo de resolución son los siguientes:

- **Clase P**: el problema puede ser resuelto en tiempo polinómico.
- **Clase NP**: se puede verificar la validad de la resolución de un problema en tiempo polinomial.

- **Clase NP-complete**: es un subconjunto de NP en el que cada problema de tipo NP puede ser reducido a un problema del subconjunto NP-complete en tiempo polinomial.
- **Clase NP-hard**: es un conjunto que incluye a todos los problemas de tipo NP-complete donde la resolución de los mismos puede ser de mayor complejidad que en la de los problemas de tipo NP.

Estos tipos de complejidades se pueden resumir en el siguiente esquema:

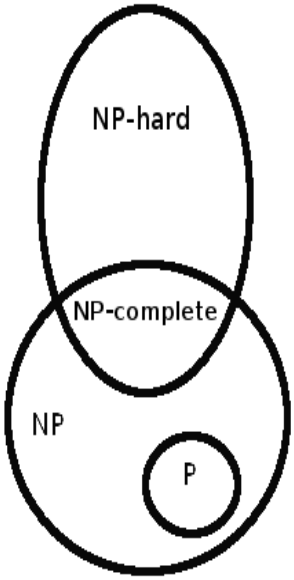

Figura 1: Tipos de complejidades computacionales

De los tipos de complejidades computacionales descritos, la **complejidad máxima para cualquier problema de planificación es la máxima complejidad posible, es decir, es NP-hard**.[39]

5.2.7 Planificadores utilizados

5.2.7.1 Metric-FF

Metric-FF[40] es un **planificador independiente de dominio** que **permite operar** con sentencias de *PDDL 2.1* y **con sentencias numéricas con dígitos**. Este planificador es una extensión del planificador *FF* para incluir construcciones numéricas[41] que resuelvan tareas lineales (es decir, que el valor de las variables numéricas pueda ser modificado mediante los operadores de suma y resta, excluyendo otras operaciones como la multiplicación o la división). Está implementado en *C*, ambos desarrollados por Joerg Hoffmann bajo la licencia de *GNU General Public License*. *Metric-FF* fue uno de los planificadores presentados en la tercera edición de la *IPC* obteniendo el mejor desempeño de todos los planificadores en el track

numérico[42]. Existen a su vez otras versiones de *FF*, como pueden ser *Conformant-FF*, *Contingent-FF*[43] o *FF-X*.

Para poder buscar una solución,el planificador necesita un árbol de estados, un valor heurístico para cada estado del árbol de estados y un algoritmo de búsqueda que permita buscar soluciones en dicho árbol. Este árbol se configura mediante una relajación de las condiciones del problema inicial y un intento de buscar una solución al problema relajado, por ejemplo mediante la conservación de los estados eliminados por las acciones. Lo que diferencia a esta extensión llamada *Metric-FF* del planificador *FF* original es que también utiliza también técnicas de relajación de las cláusulas para los dominios numéricos con dígitos. Estas técnicas incluyen en los dominios numéricos la relajación de decrementos de las variables numéricas, puesto que en estos dominios estas variables cuando aparecen en la parte de los efectos de los operadores, suelen aparecen en calidad de variación de sus valores (incrementos o decrementos de los mismos) en lugar de por eliminación de las variables en sí. Sin embargo, para que esta relajación del problema sea válida se debe comprobar antes que siempre es preferible tener valores elevados (a estos problemas se les llama *monotónicos*) para poder alcanzar la solución, ya que de no ser así se podría incluso perder la solución al intentar relajar el problema.

El proceso de búsqueda de la solución se realiza utilizando el algoritmo de búsqueda en escalada o *Enforced Hill Climbing* (*EHC*). Este algoritmo es una variación del algoritmo de búsqueda de colina o *Hill Climbing*. Antes de explicar la particularidad del algoritmo utilizado en *Metric-FF* respecto al algoritmo de *Hill climbing*, se va a explicar cómo funciona este último.

El algoritmo de *Hill climbing* es un algoritmo de **búsqueda local** iterativo que expande el estado inicial hasta encontrar un estado que sea la solución. Es un algoritmo de búsqueda local sin memoria que genera iterativamente los sucesores del último estado expandido. De estos sucesores, el mejor de estos sucesores es el que se expande. En caso de que el sucesor no sea mejor que el estado padre, la búsqueda falla.

La diferencia entre este algoritmo y *Enforced Hill Climbing* reside en que mientras que en *Hill Climbing* **la mejora iterativa** se hace mediante la selección de uno de los mejores sucesores del estado actual, en ***Enforced Hill Climbing* se utiliza una búsqueda en anchura** o *breadth first seach* para encontrar el primer sucesor cuya evaluación heurística sea mejor que la del nodo actual. El *problema* que tienen

ambos algoritmos es que al ser una búsqueda local, este algoritmo **falla en caso de que los valores heurísticos de los sucesores se topen con mínimos locales**, lo que se da si todo sucesor tiene peor valor heurístico, por lo que el algoritmo no hallaría ninguna de las rutas por las que se llega a la solución óptima.

Respecto a las fecha primera versión del planificador del que deriva, *FF*, fue publicada en 2001 y la versión de *Metric-FF* utilizada en este trabajo es la versión 2.1 (la última versión) publicada a fecha 04/2017.

5.2.7.2 Fast Downward

Fast Downward[44] es un planificador que puede solucionar problemas expresados en *PDDL*. Al igual que ocurre con *FF*, *Fast Downward* es un planificador progresivo. La heurística, que utiliza descomposiciones de las tareas, se denomina heurística de grafo casual. Entre sus logros, se encuentra el ser el planificador ganador del track clásico de la cuarta edición de la *IPC*.

Respecto a la versión utilizada, se ha utilizado la versión presentada en la *IPC* de 2011.

5.2.7.3 LPG

El planificador *Local Search Planner* (*LPG*)[45] es, al igual que los planificadores anteriores, un planificador independiente de dominio y su primera versión salió a la luz en el año 2003. Este planificador utiliza varias heurísticas basadas en una función de evaluación parametizada donde los parámetros son evaluados dinámicamente. Es un **planificador incremental que produce planes basados en varios criterios de calidad**. El núcleo de este planificador está basado en **búsquedas estocásticas locales** y en una representación basada en grafos llamada Grafos de Acción Temporales (*TA-graphs*).

Para realizar la búsqueda local, *LPG* utiliza el algoritmo *Walkplan* con el fin de encontrar el grafo de solución. Para ello, primeramente habrá que generar un grafo de acciones (*A-graph*) inicial, siendo un grafo de acción un subgrafo de un grafo de planificación conteniendo los nodos de acciones iniciales y los nodos de acciones finales (que son el último paso antes de alcanzar la solución). Si un nodo de acción del grafo de planificación está en este grafo de acción entonces también los nodos correspondientes a las precondiciones y los efectos positivos de estas acciones deben estar en este nuevo grafo de acciones.

Este *A-graph* inicial podrá ser generado de diversas maneras, como puede ser generando un grafo vacío o inicializado aleatoriamente (este segundo caso explica por qué este planificador es estocástico). Una vez se ha generado este grafo inicial, se pueden añadir o quitar nodos de acción hasta llegar a encontrar una solución.

La versión utilizada en las pruebas es la versión de 2004, *LPG-td*, ya que esta versión introduce la mejora respecto a la versión 1.2 de poder resolver problemas numéricos.

5.3 Modificadores automáticos de dominios y problemas

En esta sección se van a explicar en qué consisten algunos de los modificadores automáticos de dominios independientes de dominio. La mayor parte de estos modificadores utilizan exclusivamente una única estrategia genérica para modificar ese dominio sin alternarla con otras posibilidades. Las estrategias más frecuentes suelen ser: la representación de los dominios y problemas en formato numérico con dígitos o con proposiciones o el aglutinamiento de varios operadores en macrooperadores.

5.3.1 Generadores de macrooperadores

Los macrooperadores son la unión de varios operadores en un solo operador que tienen por **objetivo servir como atajo a la resolución de problemas al poder realizar las acciones equivalentes a varios operadores con un solo operador**. Estos macrooperadores **pueden ser perjudiciales** para la resolución de problemas de planificación ya que **tienden a incrementar el factor de ramificación** durante la búsqueda de la solución, puesto que por lo general los macrooperadores suelen incrementar el número de operadores aplicable, incrementando a su vez la memoria consumida por el planificador. Por ello la utilización de los macrooperadores no siempre mejora los tiempos de búsqueda de la solución por parte del planificador ni la calidad de la misma (entendida como el número de pasos para llegar a la solución o el valor de la métrica que haya que minimizar o maximizar según sea definido en el problema). Deberán ser los propios algoritmos generadores de estos macrooperadores quienes deban tener en cuenta estos contrapuntos y, en caso de que el generador esté bien programado, no genere macrooperadores en aquellos problemas en los que los inconvenientes no compensen la mejora que aportan dichos cambios y por tanto devuelvan los dominios como se encontraban originalmente. Este problema recibe el nombre de **"el problema de la utilidad"**.

Respecto a la longitud de estos macrooperadores, trabajos recientes han demostrado que **es mejor generar pocos macrooperadores y con un tamaño pequeño**

que generar muchos macrooperadores y con un tamaño grande. Puesto que las diferentes versiones del dominio utilizado para las pruebas empíricas que aparecen en la sección de ola solución propuesta incluyen varias versiones con macrooperadores de los dos tipos, esta afirmación se pondrá a prueba con los resultados obtenidos para este caso concreto (aun sabiendo que de un caso tan específico no es posible sacar conclusiones más que el servir como un apoyo más de dicha afirmación y no una generalización).

A continuación se incluye la descripción de algunos ejemplos de generadores automáticos de macrooperadores, siendo todos ellos independientes de dominio.

5.3.1.1 Planning Task Transformer

El generador de macrooperadores *Planning Task Transformer* (PTT)[46] es un generador *offline* que se basa en la **búsqueda de enredos** (*entanglements*) -es decir, la relación de los operadores y los predicados- **que pueden luego llegar a convertirse en macrooperadores**. A su vez estos macrooperadores son aprendidos a partir de los planes, pues es en estos planes donde se puede observar por ejemplo si dos acciones que actualmente no son adyacentes pueden llegar a serlo mediante permutaciones del órden de ejecución de los operadores. Los macrooperadores se generan según varios criterios, como puede ser el tamaño del macrooperador que se va a generar, el número de veces que se sucede la secuencia de operadores encapsulados o el número de macrooperadores totales generados, aunque PTT no tiene un límite fijo de macrooperadores que pueden llegar a generarse. Por otro lado, la eliminación de todo los operadores encapsulados por estos macrooperadores generado con PTT puede implicar la pérdida de la completitud. Sin embargo, las pruebas realizadas por los desarrolladores de este generador demuestran que por lo menos en los dominios presentados a las competiciones de la *IPC*, esta eliminación haría perder la solvabilidad muy raramente.

Respecto a los enredos en los que se basan estos macroopedores, en PTT se distinguen los **enredos externos** y los **enredos internos**.

Los enredos externos capturan relaciones causales entre los predicados iniciales o los predicados meta y se utilizan para podar instancias poco prometedoras de los operadores. Estos enredos se deducen a partir de los planes de los problemas y uno de los parámetros que se introduce como entrada para determinar cuál de los enredos puede pasar a convertirse en un macrooperador es el porcentaje de veces que dicho enredo no se respeta en alguno de los planes, y se denomina *flaw ratio*. **Encontrar**

estos enredos tiene una complejidad de *PSPACE-complete*, por ello estos enredos se deducen mediante algoritmos de aproximación que tienen menor complejidad, teniendo como contrapunto que la transformación de estos enredos en macrooperadores que sustituyan a los operadores iniciales generalmente no conserva la completitud. Sin embargo, los enredos no siempre se traducen en la generación de macrooperadores. El resultado final que se obtiene es que *PTT* conserva la completitud debido a que sólo se añaden un subconjunto de los macrooperadores derivados de los enredos externos que permiten que ésta se mantenga.

Los enredos internos se producen entre dos operadores en el que uno de ellos produce un predicado que es requerido (o consumido) por un segundo operador. Este tipo de enredos también se denominan enredos por sucesión o por predecesión (en función de si se está refiriendo al sucesor del productor o al predecesor del consumidor).

5.3.1.2 Marvin

Marvin es un planificador automático desarrollado a partir de *FF*, lo que queda patente en cosas como que comparten entre sí el uso de EHC como algoritmo de búsqueda o que ambos realicen el proceso de búsqueda hacia adelante. La diferencia más importante es que ***Marvin* incluye una generación de macrooperadores** para la posterior búsqueda de la solución, mientras que *FF* realiza esta búsqueda sin realizar este paso previo. En las siguientes líneas se va a intentar explicar cómo funciona dicho módulo del planificador.

Este planificador **utiliza los macrooperadores como medio para intentar escapar de los *plateaus***, ya que estos son los responsables de que los algoritmos de búsqueda local se queden estancados y pierdan la mayor parte del tiempo de búsqueda del algoritmo. Como *Marvin* utiliza el algoritmo de búsqueda local *EHC*, evitar caer en estos *plateaus* es una tarea fundamental para reducir los tiempos de búsqueda.

Los macrooperadores son seleccionados en función de las veces que se repita el mismo *plateau* con diferentes problemas. También influye en el proceso de selección de macrooperadores el tamaño de dicho *plateau*, habiendo una correspondencia directa entre la probabilidad de que un macrooperador sea seleccionado y el tamaño del *plateau* en cuestión. Esto quiere decir que al contrario que en el caso de *PTT*, con *Marvin* se potencian los macrooperadores de mayor tamaño.

Por último, es importante comprobar que los macrooperadores utilizados tienen el mínimo impacto en la longitud del plan: no debe existir una secuencia de operadores iniciales que conlleve al resultado en menos pasos que la aplicación de dicho macrooperador.

5.3.2 Partición de operadores

Como se ha comentado el la sección anterior, **el beneficio de la inclusión de macrooperadores en la planificación automática está en disputa con otras pérdidas que influyen en la calidad y en el tiempo de búsqueda**. A esta disputa entre pérdidas y ganancias se le denomina ***el problema de la utilidad***.

La partición de operadores parte de esta misma base y tiene el enfoque complementario: algunos operadores al ser fragmentados permitirán obtener beneficios en cuanto a la calidad y al tiempo de búsqueda del planificador debidos a la disminución del tamaño de la interfaz de los operadores. En contraste con las técnica de generación de macrooperadores, la segmentación de operadores ha sido mucho menos estudiada entre la comunidad científica.

La modificación de segmentar las acciones en otras más pequeñas es un tipo de modificación del dominio que es muy frecuente en modificaciones realizadas a mano sobre dominios. La principal ventaja de esta técnica es la redución de los operadores básicos o *ground operators*. Por ejemplo, dada una acción $a[X]$ con un tamaño de interfaz $|X|$, su segmentación creará los operadores $a_1[X_1], ..., a_k[X_k]$. Si por ejemplo cada $x \in X$ puede ser instanciada con 100 objetos, $|X| = 3$ y $|X_i| = 1$ (el tamaño de la interfaz para cada operador tras la segmentación), en el primer caso el número de posibles instancias es $100^3 = 1.000.000$ mientras que en el caso segmentado sería de $100 * 3 = 300$.

Para que una separación de operadores sea válida, se deben cumplir las siguientes condiciones:

- Los operadores $a_i[X], ..., a_k[X]$ deben de ejecutarse en bloque (sin que se ejecuten otras acciones de por medio) y respetando el órden en que lo hacían los operadores iniciales.
- Los operadores $a_i[X], ..., a_k[X]$ deben estar configurados de tal manera que al ser ejecutados se asegure que el o los predicados utilizados por estas acciones se mantenga a lo largo de la ejecución. Esto quiere decir que por ejemplo no sea posible que cambie uno de los predicados y por tanto las acciones segmentadas den un resultado distinto de la ejecución de la acción original.

Un ejemplo de software de partición de operadores es *Automatic Action Schema Splitting*[47].

5.3.3 Reprentación numérica con dígitos

Este tipo de modificadores se basa en detectar conjuntos de predicados declarados en el problema que pueden ser agrupados y usados indistintamente, por lo que se puede

identificasect cada uno de estos predicados por la declaración de un único predicado numérico que sustituya a estos predicados.

La funciones numéricas con dígitos fueron introducidas en *PDDL* en su versión 2.1, sin embargo, no todos los planificadores desarrollados en los últimos años aceptan este tipo de representación. De los planificadore que se han utilizado para las pruebas, *Metric-FF* y *LPG-td* aceptan este tipo de representación, a diferencia de *FastDownward* que no es capaz de interpretar funciones numéricas.

Entre las pruebas realizadas en este estudio, se encuentran los dominios y problemas obtenidos tras la ejecución de un modificador automático de este tipo desarrollado en la Universidad Carlos III de Madrid[48].

5.3.4 Representación numérica con proposiciones

Esta representación se basa en el mismo principio que la representación numérica con dígitos: en determinados problemas, existen predicados que pueden perder su identidad y ser cuantificados en función del número de ellos que se hayan definido. La principal diferencia entre ambos tipos de cambio de representaciones es que en este caso la representación numérica no es con dígitos sino con proposiciones.

Entre las pruebas realizadas en este estudio, se encuentra un tipo de modificador de la representación de problemas que realiza este tipo de transformaciones: *Baggy*[49].

Este modificador transforma la representación de dominios y problemas en *PDDL* en una segunda que reduce el tamaño del espacio de estados, eliminando a su vez la explosión combinatoria. Los pasos que se deben seguir para resolver un problema utilizando *Baggy* son los siguientes:

1. Determinar qué elementos pueden ser agrupados y **reformulación** de la representación de *PDDL*.
2. **Resolución** de la planificación con la nueva representación.

3. **Transformación** del problema del espacio de estados nuevos al espacio de estados original.

Una ventaja de este tipo de modificaciones respecto a las modificaciones que incorporan funciones numéricas con dígitos es que en este caso la nueva representación generada puede ser resuelto por cualquier planificador que pueda interpretar problemas en *PDDL*.

5.3.5 Dominios utilizados

Todos los dominios y problemas utilizados están escritos en *PDDL* y consisten en modificaciones del **dominio original de Child-Snack utilizado en la *IPC* de 2014[50]**. Los dominios utilizados para las pruebas de la inmensa mayoría de los trabajos de planificación automática citados en este documento provienen también de esta competición, en la cual hay varios dominios que se repiten o se actualizan entre las distintas ediciones, por lo cual se cree que la utilización exclusiva de dominios pertenecientes a la *IPC* puede permitir que otros investigadores puedan problematizar y comparar los resultados obtenidos en este trabajo con otras investigaciones con mayor facilidad, ya que por lo que se ha dicho se comparte el marco de problemas utilizado con muchas de las investigaciones recientes en este área.

6 Desarrollo del trabajo

Para el desarrollo de este proyecto **se ha utilizado** software de terceros que en todas las etapas del mismo ha sido **exclusivamente software libre y gratuito** motivo por el cual no va a ser señalada esta característica para cada una de las herramientas utilizadas que será citada a continuación. En esta sección se puede encontrar la descripción de todos los dominios utilizados así como de las herramientas para su desarrollo y posterior testeo.

6.1 Herramientas de desarrollo utilizadas

En esta sección se detallan todas las herramientas tecnológicas utilizadas para el desarrollo de este proyecto. Como el hardware utilizado afecta exclusivamente al desempeño de las pruebas funcionales, estas características se incluirán en la sección de resultados obtenidos.

6.1.1 Sistema operativo

El sistema operativo utilizado en cada una de las etapas del proceso ha sido **Debian GNU-Linux** versión 8.7. Este sistema operativo ha sido elegido para el proyecto ya que por una parte algunos de los programas de terceros desarrolladores están pensados para ser utilizados en sistemas operativos GNU/Linux y por otra se tenía cierta familiaridad previa con el mismo. Este sistema operativo puede descargarse en la siguiente página web: www.debian.org [última vez visitada: 18/05/2017].

6.2 Edición de código

El código ha sido editado y desarrollado mediante el editor de texto plano **Gedit**, que es un editor el cual viene incluído en el paquete de instalación básico de muchas distribuciones de GNU-Linux. Puede descargarse en la siguiente página web: www.wiki.gnome.org/Apps/Gedit [última vez visitada: 18/05/2017]. Este editor ha sido utilizado por su sencillez. Sin embargo, para proyectos más complejos puede ser mejor utilizar otros editores como Vim[1] o Emacs[2], ambos de software libre y gratuito.

6.3 Control de versiones

Para el control de versiones de código se ha utilizado un repositorio de **Git** alojado en GitHub con un único repositorio para alojar todo el código utilizado así como otro repositorio para las distintas versiones de la memoria.

[1] www.vim.org
[2] https://www.gnu.org/software/emacs/

De esta manera, en caso de pérdida o avería del dispositivo con el que se ha desarrollado este proyecto, existía una copia de seguridad de todo el trabajo realizado en GitHub.

Al haberse escrito esta memoria en un formato que antes de ser transformado a formato *PDF* es texto plano (en contraposición con el formato de texto enriquecido de editores como LibreOffice Writer), en lugar de usar un repositorio de documentos como Alfresco[3], se ha podido crear otro repositorio de Git que detectase los cambios entre las distintas versiones de la memoria.

6.4 Edición de la memoria

La memoria ha sido desarrollada en **LaTeX (latex)** y en este caso por comodidad ha sido escrita en el editor en línea de Sharelatex. Esto es debido únicamente a que a diferencia de lo que ocurre con el editor Gedit estándar (es decir, sin añadir ningún complemento), en este editor en línea es posible comprobar los errores de sintaxis antes de haber compilado el archivo así como el hecho de que la propia interfaz permite visualizar en una única ventana el documento PDF que se va a generar y ver la correspondencia entre la salida en dicho PDF con la línea exacta del archivo de latex que la genera.

Una vez modificados los documentos en Sharelatex, estos eran pasados a Git de forma manual: el proyecto de Sharelatex era descargado y después subido a Git como *commit* con los cambios que detectase.

6.5 Scripts para la traducción de problemas manuales

La mayoría de las versiones de los dominios y problemas realizados a mano han sido realizados mediante scripts que tuviesen como parámetro de entrada el problema original y como archivo de salida el problema en la versión modificada a mano correspondiente. Estos scripts se han realizado en el lenguaje de programación **Python**.

6.6 Scripts para la generación de problemas automáticos

Los scripts de terceros utilizados para la transformación automática de dominios generan un dominio por cada ejecución. Con el fin de automatizar la transformación de todos los problemas, se han creado varios un script en **Bash** para cada uno de los modificadores para así poder obtener todos los problemas modificados a partir de aquellos que se encuentren en un directorio especificado sin tener que ejecutar a

[3] www.alfresco.com

mano cada uno de estos scripts de transformación por cada uno de los problemas a transformar.

6.7 Scripts para la automatización de pruebas

Se ha realizado un script en **Bash** que permite obtener los tiempos de ejecución y la calidad de la solución de la planificación automática de la versión del dominio y los problemas que se introduzcan como parámetros para los planificadores *Metric-FF* y *LPG-td*. En la sección de evaluación de resultados se concreta exactamente en qué consisten estas pruebas.

6.8 Descripción de los dominios

En esta sección se van a describir todos los dominios utilizados en el proyecto como entrada de los distintos planificadores utilizados en la fase de obtención de tiempos y de calidad de las soluciones para su ulterior análisis.

Todas las versiones de dominios utilizados provienen de la edición de 2014 de la *IPC*. Todos los dominios analizado son modificaciones del dominio de Childsnack de esta edición. Este tipo de modificaciones, sin embargo, no son exclusivas de un dominio concreto; la correlación entre la modificación de las representaciones y la calidad de las soluciones obtenidas ya ha sido estudiada por otros investigadores[51][52].

Las modificaciones realizadas al dominio de *Childsnack* han sido realizadas tanto a mano como mediante software de terceros independiente de dominio.

En el caso de que uno de los dominios derive de otro, únicamente se va a explicar en el dominio derivado los motivos y los cambios respecto al dominio original. Por último, en todos ellos se explicará si los nuevos dominios son completos y si se pierde la optimalidad respecto al dominio original.

El resúmen de los cambios realizados en cada versión se puede encontrar a continuación:

- **Versión original**: versión sin modificar de *Childsnack*.
- **Versión 01**: utiliza macrooperadores. Junta los operadores `move_tray` con `serve_sandwich` y con `serve_sandwich_no_gluten` , elimina el operador `move_tray` e incluye el operador `move_tray_kitchen` para que las bandejas portátiles no puedan moverse entre las mesas.

- **Versión 02**: cambia el operador `make_sandwich` para que no acepte como entrada componente y pan sin gluten para que así no lo etiquete como que tuviesen gluten cuando en realidad no lo tienen.
- **Versión 03**: utiliza macrooperadores. Permanecen únicamente los siguientes operadores:
  `make_sandwich_move_tray_serve_sandwich_no_gluten_move_tray_kitchen` ,
  `make_sandwich_move_tray_serve_sandwich_gluten_move_tray_kitchen`
  y `move_tray_kitchen` .
- **Versión 05**: es la versión numérica manual con proposiciones.
- **Versión 08**: es la versión numérica manual con dígitos.
- **Versión 09**: aglutina varios predicados en uno. Se elimina el predicado `(waiting ?c ?p)` para pasar a definir la localización del niño con `(not_allergic_gluten ?c ?p)` y `(allergic_gluten ?c ?p)` .
- **Versión 11**: cambia predicados negativos por la negación de estos predicados en positivo. Cambia los predicados `(not_allergic_gluten ?c)` y `(served ?c)` por `(not (allergic_gluten ?c)` y `(not (waiting ?c))` respectivamente.
- **Versión 12**: versión en la que se segmenta la ejecución en distintas etapas, sin permitir avanzar hasta que se hayan completado las etapas anteriores.
- **Versión PTT**: es la misma que la versión original, porque este software ha sido incapaz de encontrar mejoras a la representación.
- **Versión numérica con proposiciones**: generada automáticamente por *Baggy*.
- **Versión numérica con dígitos**: generada automáticamente. El dominio varía respecto a la versión 08 en que en esta versión se incluyen dos operadores para el operador `serve_sandwich` de la versión 08. La representación de los problemas es la misma en ambas versiones.

6.9 Versiones del dominio Childsnack

A continuación, se encuentra la descripción de todas las versiones del problema de *Childsnack* que han sido utilizadas como entrada de los distintos planificadores para la obtención de resultados que se encuentra en la sección posterior a esta.

6.9.1 Versión original

El dominio original recibe el nombre de *Childsnack* y fue presentado por primera vez en la edición del año 2014 de la *IPC* (International Planning Competition). En

este dominio se pretende organizar la creación y distribución de sándwiches a partir de unos componentes con y sin gluten para un grupo de niños donde parte de ellos puede que sean alérgicos al gluten, por lo que en este caso los dos componentes de los sándwiches deberán necesariamente no tener gluten. Por otro lado, en el caso de los niños que no son alérgicos al gluten será indiferente si alguno o todos los componentes del sándwich contienen o no gluten.

Los sándwiches se definen con el predicado `(at_kitchen_sandwich ?s - sandwich)` y constan de dos componentes: el pan y el contenido, que son definidos mediante los predicados `(at_kitchen_bread ?b - bread-portion)` y `(at_kitchen_content ?c - content-portion)` . Todos los sándwiches se hacen en la cocina (que se define a su vez mediante la constante `kitchen`) y únicamente se gasta un trozo de pan y de componente por cada sándwich. En la definición original del dominio, únicamente se puede definir la cocina como localización de los componentes del sándwich, a diferencia de lo que ocurre con las bandejas portátiles, los sándwiches y los niños. Una vez se ha creado el sándwich, los componentes utilizados son eliminados de la lista de hechos (no son reutilizables).

Como ya se ha comentado, los componentes de los sándwiches así como el sándwich final podrá tener o no gluten. En caso de que alguno de estos elementos tenga gluten, no será necesario añadir ningún predicado adicional. Por contra, para señalar que uno de estos elementos no contiene gluten se utilizará el predicado correspondiente de los siguientes: `(no_gluten_bread ?b - bread-portion)` , `(no_gluten_content ?c - content-portion)` , `(no_gluten_sandwich ?s - sandwich)` .

Los sándwiches que se crean en la cocina pueden después ser cargados y transportados utilizando bandejas portátiles. Estas bandejas portátiles no tienen ninguna restricción en cuanto a cantidad de sándwiches que pueden transportar y permiten que los sándwiches puedan ser transportados libremente entre la cocina y las distintas localizaciones de los niños, permitiendo a su vez que tanto el origen como el destino de la bandeja sean las distintas posiciones de los niños. Las bandejas se declaran mediante el predicado `(at ?t - tray ?p - place)` , que indica la localización de cada una de ellas.

En caso de que coincida la localización de la bandeja con la cocina, los sándwiches que queden creados en la cocina pueden ser cargados en la bandeja, para ello se declara el predicado `(ontray ?s - sandwich ?t - tray)` . A su vez, en caso de coincidir la localización de la bandeja con la de alguno de los niños que aún no

haya sido servido y la bandeja esté cargada con algún sándwich que se adapte a sus necesidades en cuanto al gluten se refiere, este sándwich puede ser eliminado de la bandeja y servido al niño.

Los niños que no han sido servidos y que se espera que tras la resolución del problema sean servidos son declarados mediante el predicado `(waiting ?c - child ?p - place)` cuando están esperando y con el predicado `(served ?c - child)` cuando ya han sido servidos.

La dificultad principal de este dominio se basa en la limitación de componentes sin gluten que son necesarios para la creación de sándwiches para los niños alérgicos al gluten, ya que en el caso de los no alérgicos cualquier sándwich servido resolvería el problema. Esto podría implicar por ejemplo que se utilizase erróneamente un componente sin gluten con otro componente con gluten que podría ser necesario para crear otro sándwich sin gluten. Sin embargo, en otra situación podría ocurrir que dicha combinación fuera correcta ya que no quedase otra opción para servir al niño que queda que no es alérgico al gluten. A su vez, no se le puede servir a un niño no alérgico al gluten un sándwich etiquetado como sin gluten, por lo que la solución de hacer tantos sándwiches sin gluten como permita el dominio y después si hace falta distribuir esos sándwiches entre los no alérgicos se descarta al no ser una solución válida para cualquier problema de entrada.

Al contrario que ocurre con los componentes, para cada niño es necesario especificar tanto si son como si no son alérgicos al gluten mediante los predicados `(allergic_gluten ?c - child)` y `(not_allergic_gluten ?c - child)` .

A este problema hay que sumarle que ni el número de niños definidos en el problema ni el número de sándwiches máximos que se pueden realizar, definidos mediante el predicado `(notexist ?s - sandwich)` , tiene por qué coincidir con el número y la composición de alérgicos de niños definidos en la meta como que tienen que ser servidos, por lo que incluso aunque se incluyese conocimiento del dominio en la resolución del problema por parte del planificador, el problema seguiría sin ser trivial. De todas maneras, aunque el predicado `(notexist ?s - sandwich)` se puede utilizar para limitar el número máximo de sándwiches que se pueden realizar, su verdadera función es que exista una correlación de a uno en los sándwiches creados.

Tras haber explicado la idea general del dominio así como sus dificultades a la hora de resolver los problemas, se va a explicar la entrada y salida de cada una de las

acciones del dominio así como la estructura general que deben tener los problemas de entrada.

Las acciones del dominio original son:

- **make_sandwich_no_gluten (porción de pan sin gluten, porción de contenido sin gluten)**
  - **Precondiciones**: que exista al menos un contenido y un trozo de pan donde ambos sean sin gluten y haya al menos un hecho de no existencia de sandwich.
  - **Efectos**: eliminación de los dos componentes utilizados y del hecho de no existencia de sándwich y creación de un sandwich sin gluten.
- **make_sandwich (porción de pan, porción de contenido)**
  - **Precondiciones**: que exista al menos un contenido y un trozo de pan con o sin gluten y haya al menos un hecho de no existencia de sándwich.
  - **Efectos**: eliminación de los componentes utilizados y del hecho de no existencia de sándwich y creación de un sandwich etiquetado como con gluten (aún en el caso de que ambos elementos utilizados no tengan gluten).
- **put_on_tray (sándwich, bandeja portátil)**
  - **Precondiciones**: que exista al menos un sándwich en la cocina y que al menos una de las bandejas portátiles esté en la cocina.
  - **Efectos**: carga uno de los sándwiches en una de las bandejas portátiles que esté en la cocina.
- **serve_sandwich_no_gluten (sándwich sin gluten, bandeja portátil, niño, posición)**
  - **Precondiciones**: que al menos una de las bandejas portátiles esté en la misma posición que uno de los niños, que la bandeja que cumpla la precondición anterior tenga al menos un sándwich sin gluten, que un niño no alérgico al gluten esté en la misma localización que la bandeja que cumple las precondiciones anteriores y que este mismo niño esté esperando un sándwich.
  - **Efectos**: el niño pasa a estar servido y se elimina el sándwich utilizado en la operación.
- **serve_sandwich (sándwich, bandeja portátil, niño, posición)**

  - **Precondiciones**: que al menos una de las bandejas portátiles esté en la misma posición que uno de los niños, que la bandeja que cumpla la precondición anterior tenga al menos un sándwich con o sin gluten, que un niño no alérgico al gluten esté en la misma localización que la bandeja que cumple las precondiciones anteriores y que este mismo niño esté esperando un sándwich.
  - **Efectos**: el niño pasa a estar servido y se elimina el sándwich utilizado en la operación.

- **move_tray (bandeja portátil, posición inicial, posición final)**
  - **Precondiciones**: que la bandeja esté en la localización inicial.
  - **Efectos**: mueve la bandeja de su localización inicial a una nueva localización.

En el problema se podrán definir los siguientes elementos:

- Bandejas portátiles, con la localización inicial de cada uno de ellos.
- Panes que se encuentran en la cocina.
- Contenidos (de los sándwiches) que se encuentran en la cocina.
- Especificación de los trozos de pan que no llevan gluten.
- Especificación de los contenidos que no llevan gluten.
- Especificación de los niños alérgicos al gluten.
- Especificación de los niños que no son alérgicos al gluten.
- Especificación de los niños que están esperando que les sirvan un sándwich y su localización.
- Especificación de los niños que tienen que ser servidos para resolver el problema.

6.9.2 Versiones modificadas a mano

En esta sección se describen los dominios que han sido generados a mano, en contraposición con los dominios generados mediante software de terceros que serán explicados en la sección que se encuentra a continuación de esta.

6.9.2.1 Versión 01

Mientras que la definición de los problemas en este dominio no varía respecto a la definición original, en la definición del dominio se han añadido macrooperadores de las acciones `move_tray` con las acciones `serve_sandwich` y `serve_sandwich_no_gluten` , quedando por tanto los macrooperadores `move_tray_serve_sandwich` y `move_tray_serve_sandwich_no_gluten` . La inclusión de macrooperadores también ha implicado la inclusión de la función numérica `total_cost` , inicializada a cero en cada uno de los problemas de entrada. Esta función es utilizada tanto en esta versión como en la versión 03 y en ambas tiene exactamente la misma razón de ser: aumenta su valor al ejecutar los operadores en tantas unidades como operadores del problema inicial esté sustituyendo, teniendo entre las métricas de sus problemas la reducción de su coste. Esto implica que los dos operadores utilizados en este dominio incrementan dicho valor en dos unidades mientras que el resto de operadores al ser los operadores iniciales incrementan su valor en una unidad. El motivo de la inclusión de esta función es que por un lado de esta manera al obtener el coste total de la solución no se obtienen costes artificialmente bajos (como sucedería si no coincidiese e coste de un macrooperador que el de los operadores contenidos) y por otro, al introducir la métrica de su reducción, los planificadores podrán tener esto en cuenta durante su planificación con el fin de obtener soluciones de menor coste.

Respecto a los macrooperadores mencionados en el párrafo anterior, estos tienen como fin unir la acción de mover la bandeja con la acción de servir el sándwich ya que viendo la resolución de problemas se puede comprobar con facilidad que la solución óptima tras mover la bandeja a una mesa es servir al menos un sándwich, ya que el objetivo de dicho desplazamiento es exclusivamente servir dicho sándwich que además es justo la siguiente acción a realizar, por lo que intuitivamente tiene sentido unir ambas acciones, así como añadir una acción para mover la bandeja a la cocina para volver a cargarlo.

Los nuevos macrooperadores incrementan el número de parámetros de las precondiciones del operador, pasando de dos a cuatro. A su vez, implica aumentar el factor de ramificación, ya que se ha aumentado en dos el número de operadores disponibles.

En lo que concierne tanto a la completitud y a la optimalidad, siempre que se añadan macro-operadores sin eliminar los operadores originales ambas condiciones

se conservarán, puesto que en el peor de los casos en los que no se pueda obtener la solución óptima o completa utilizando los macro-operadores, siempre estará la opción de utilizar los operadores originales. En este caso, si se eliminasen los operadores originales `move_tray` , `serve_sandwich` y `serve_sandwich_no_gluten` contenido en los nuevos macro-operadores y se añadiese a su vez el operador `move_tray_kitchen` , el dominio sería completo puesto que todas las soluciones del dominio original están contenidas en el nuevo dominio. Sin embargo, este dominio no sería óptimo puesto que al perder el operador `move_tray` , en todos los problemas en los que los niños estuvieran en varias localizaciones, sería necesario que una vez servido a el/los niño/s de la primera localización, se volviese necesariamente a la cocina para poder volver a mover la bandeja y servir a el/los siguiente/s niño/s, perdiendo por tanto la solución óptima al tener que realizar el movimiento innecesario respecto al dominio original de tener que volver a la cocina para poder servir el/los siguiente/s sándwich/es. Para que quede este punto más claro, se va a mostrar un ejemplo de un problema muy sencillo en el que se puede ver claramente cómo se produce esta pérdida de la solución óptima.

Definición del problema de ejemplo:

- Pan `bread1` con gluten
- Pan `bread2` sin gluten
- Contenido `content1` con gluten
- Contenido `content2` sin gluten
- Niño `child1` alérgico al gluten
- Niño `child2` no alérgico al gluten
- Niño `child1` en table1
- Niño `child2` en table1
- bandeja `tray1` en kitchen

Solución óptima con el dominio original:

```
 0:  (MAKE_SANDWICH_NO_GLUTEN SANDW1 BREAD1 CONTENT1)
1:  (MAKE_SANDWICH SANDW2 BREAD2 CONTENT2)
2:  (MAKE_SANDWICH SANDW3 BREAD3 CONTENT3)
3:  (PUT_ON_TRAY SANDW1 TRAY1)
4:  (PUT_ON_TRAY SANDW2 TRAY1)
```

```
5:  (PUT_ON_TRAY SANDW3 TRAY1)
6:  (MOVE_TRAY TRAY1 KITCHEN TABLE1)
7:  (SERVE_SANDWICH_NO_GLUTEN SANDW1 CHILD1 TRAY1 TABLE1)
8:  (SERVE_SANDWICH SANDW2 CHILD2 TRAY1 TABLE1)
9:  (SERVE_SANDWICH SANDW3 CHILD3 TRAY1 TABLE1)
```

Solución con la versión 01 sin los operadores iniciales:

```
0:  (MAKE_SANDWICH SANDW3 BREAD3 CONTENT3)
1:  (MAKE_SANDWICH SANDW2 BREAD2 CONTENT2)
2:  (MAKE_SANDWICH SANDW1 BREAD1 CONTENT1)
3:  (PUT_ON_TRAY SANDW1 TRAY1)
4:  (PUT_ON_TRAY SANDW2 TRAY1)
5:  (PUT_ON_TRAY SANDW3 TRAY1)
6:  (MOVE_TRAY_SERVE_SANDWICH_NO_GLUTEN TRAY1 KITCHEN TABLE1 SANDW1 CHILD1)
7:  (MOVE_TRAY_KITCHEN TRAY1 TABLE1)
8:  (MOVE_TRAY_SERVE_SANDWICH TRAY1 KITCHEN TABLE1 SANDW2 CHILD2)
9:  (MOVE_TRAY_KITCHEN TRAY1 TABLE1)
10:  (MOVE_TRAY_SERVE_SANDWICH TRAY1 KITCHEN TABLE1 SANDW3 CHILD3)
```

Solución con la versión 01 con los operadores iniciales:

```
0:  (MAKE_SANDWICH_NO_GLUTEN SANDW1 BREAD1 CONTENT1)
1:  (MAKE_SANDWICH SANDW2 BREAD2 CONTENT2)
2:  (MAKE_SANDWICH SANDW3 BREAD3 CONTENT3)
3:  (PUT_ON_TRAY SANDW1 TRAY1)
4:  (PUT_ON_TRAY SANDW2 TRAY1)
5:  (PUT_ON_TRAY SANDW3 TRAY1)
6:  (MOVE_TRAY_SERVE_SANDWICH_NO_GLUTEN TRAY1 KITCHEN TABLE1 SANDW1 CHILD1)
7:  (SERVE_SANDWICH SANDW2 CHILD2 TRAY1 TABLE1)
8:  (SERVE_SANDWICH SANDW3 CHILD3 TRAY1 TABLE1)
```

Como se puede ver, estando la bandeja ya en la misma posición que `child2` y `child3` , sin los operadores iniciales necesita volver a la cocina para poder servirles, por lo que queda demostrado con este ejemplo que los operadores iniciales son necesarios para conservar la optimalidad. Por este motivo, se van a conservar dichos operadores en la versión final de esta versión.

6.9.2.2 Versión 02

La defición de los problemas no varía respecto a la definición original, por lo que en las siguientes líneas se va a explicar en qué varía este nuevo dominio respecto a la definición original. Únicamente se han modificado las precondiciones de único operador `make_sandwich` . En el dominio original, todos los sándwiches creados con este operador no son etiquetados como que no tuvieran gluten aún cuando ambos componentes utilizados no contengan gluten. Este supuesto no tiene ningún sentido, ya que implica que se están etiquetando erróneamente sándwiches que realmente no tienen gluten por haber pasado por este operador en lugar de por el operador `make_sandwich_no_gluten` , lo que implica que el propio operador permite que se generen ramas del árbol de decisión que podrían producirse a través del otro operador mencionado, por lo que están duplicados.

Para entender mejor esta modificación, se va a incluir un problema base con un grafo de búsqueda del camino que conecte el estado inicial con la solución con el dominio original y otro grafo de búsqueda con este nuevo dominio.

El problema en cuestión es el siguiente:

- 1 pan sin gluten
- 1 componente sin gluten
- 1 bandeja en la cocina
- 1 niño alérgico al gluten en table1

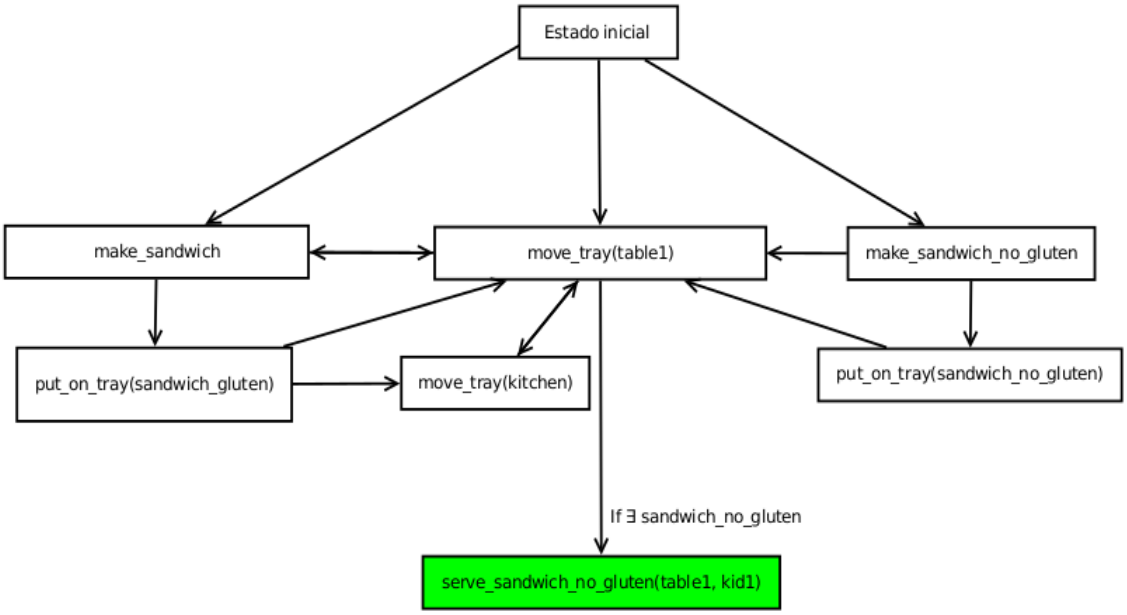

Figura 2: Grafo de búsqueda con la versión original

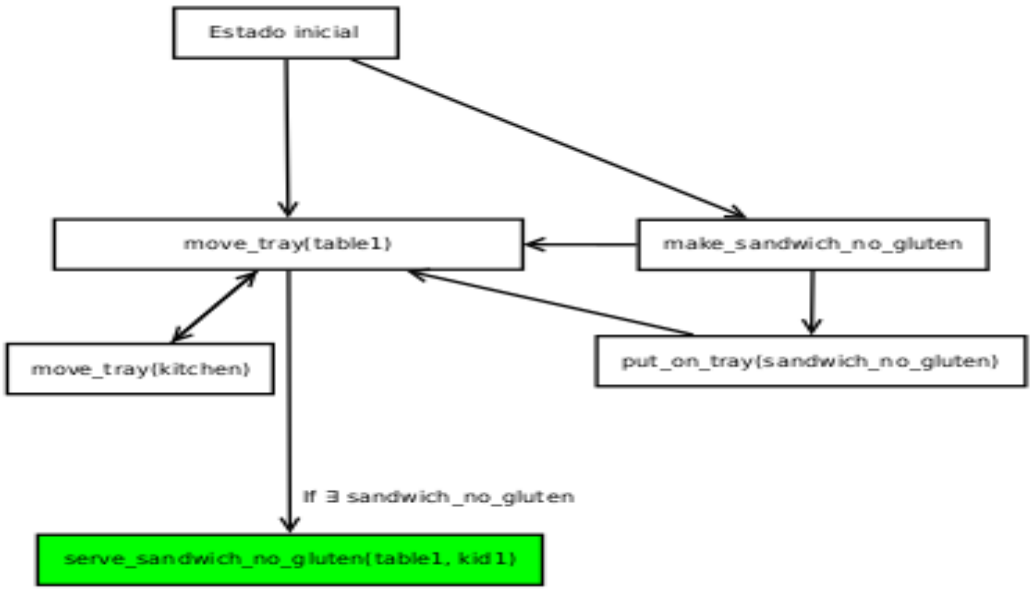

Figura 3: Grafo de búsqueda con la versión 02

Como se puede ver, el grafo de búsqueda de la versión 02 es un subconjunto del grafo de búsqueda del dominio original. Dicho de manera más genérica, este nuevo dominio sirve para podar el grafo de búsqueda original para el subconjunto de los problemas posibles en los que existen niños alérgicos al gluten y que por tanto con en el dominio original el planificador podría perder tiempo buscando en ramas del grafo de búsqueda que se sabe a priori que no van a conducir a la solución.

Esto quiere decir que este nuevo dominio es completo y óptimo, pues lo único que se ha hecho ha sido una poda de una rama del grafo de búsqueda que sólo conduce a soluciones erróneas.

6.9.2.3 Versión 03

Esta versión mantiene la definición de los problemas original, variando exclusivamente la definición del dominio. Esta versión, al igual que la versión 01 se basa en la utilización de macro-operadores, pero llevando su uso hasta el extremo. En este dominio, se han implementado los macro-operadores `make_sandwich_move_tray_serve_sandwich_move_tray_kitchen` , `make_sandwich_move_tray_serve_sandwich_no_gluten_move_tray_kitchen` y `move_tray_kitchen` que sustituirían a todos los demás operadores. Como se ha mencionado en la versión 01, al ser esta una versión que utiliza macrooperadores, hace uso también de la función numérica con dígitos `total_cost` , incrementando su coste en cuatro unidades tras la ejecución de los dos macrooperadores y en uno al ejecutar el operador `move_tray_kitchen` . La lógica subyacente a la inclusión de esta función es idéntica que en el caso de la versión 01; en la sección que se explica esta versión se puede encontrar también la explicación de la utilización de `total_cost` .

Volviendo a los operadores, los dos primeros tendrían como precondiciones que existiese al menos un componente y un trozo de pan con o sin gluten (según cuál de los dos operadores sea), que haya una bandeja en la cocina y que exista un niño alérgico o no al gluten (igualmente, según el operador que corresponda) esperando a ser servido. En caso de que se dé este supuesto, estos operadores producirían un sándwich, lo cargarían en una bandeja, lo transportarían hasta donde está el niño, servirían al niño y traerían la bandeja de vuelta a la cocina. Por otro lado, en caso de que no haya ninguna bandeja en la cocina en el estado inicial, se ejecutaría el operador `move_tray_kitchen` .

Este macrooperador reduce el árbol de búsqueda en tanto que existen únicamente tres operadores que se pueden utilizar. Sin embargo, el árbol de búsqueda contiene un estado por cada instancia posible del operador, valor directamente proporcional al número de parámetros de entrada de dicho operador, que en este caso es muy elevado. La relación entre la pérdida de estados del árbol de búsqueda debido a la disminución de operadores encapsulados en nuevos macro-operadores y la ganancia de estados en el árbol de búsqueda debido al incremento de instancias del nuevo operador se llama utilidad y será analizada en la sección de resultados de ejecución.

Si no se incluyesen los operadores originales, esta versión sería completa pero no óptima, puesto que no se podría servir a más de un niño sin tener que pasar por la cocina. Por tanto, en esta versión se conservan los operadores originales para no perder la optimalidad.

6.9.2.4 Versión 05

En esta versión, varía tanto la definición del problema como la definición del dominio. Se trata de un dominio numérico con proposiciones, en las que se definen de manera numérica el número de panes, contenidos, sándwiches con y sin gluten. Los niños no son definidos de manera numérica ya que en la definición original en la meta se especifica los niños concretos (con su respectiva localización) que deben ser servidos, por lo que si fuesen definidos de manera numérica los niños servidos con un operador como `served_childs ?num - num` se perdería esta capacidad para especificar no sólo el número sino los niños concretos dentro de todos los definidos en el problema que deben ser servidos para la correcta resolución del problema. Existe sin embargo la posibilidad de hacer contadores que incluyan a los niños pero estos planes deben después ser post-procesados. Sin ir más lejos, esto se puede lograr con *Baggy*.

Los nuevos predicados utilizados son:
`(at_kitchen_bread ?gluten - num ?no-gluten - num)` , `(at_kitchen_content ?gluten - num ?no-gluten - num)` ,
`(at_kitchen_sandwich ?gluten - num ?no-gluten - num)` y `(ontray ?gluten - num ?no-gluten - num ?t - tray)` que definen respectivamente el número de panes, contenidos y sándwiches en la cocina y los sándwiches que se encuentran en las bandejas que tengan o no gluten. A su vez, se ha añadido el predicado `(next ?minor - num ?major - num)` que sirve para incrementar y decrementar en una unidad los valores numéricos de los predicados definidos anteriormente. Los predicados del dominio original `(allergic_gluten ?c - child)` , `(not_allergic_gluten ?c - child)` , `(waiting ?c - child ?p - place)` , `(at ?t - tray ?p - place)` y `(served ?c - child)` se han mantenido.

Aunque la apariencia es totalmente distinta, la lógica de los operadores es la misma que en el caso original pero adaptada a estas nuevas definiciones numéricas con proposiciones. Únicamente ha sido necesario incluir el operador
`make_sandwich_bread_no_gluten` y `make_sandwich_content_no_gluten` que crea sándwiches con gluten con el pan o el contenido sin gluten.

Los operadores numéricos comprueban la existencia de lo que serían los parámetros necesarios en el dominio original no a través de la comprobación de la existencia de por ejemplo el nuevo parámetro `(at_kitchen_bread ?gluten - num ?no-gluten - num)` , pues siempre va a existir, sino de los valores numéricos contenidos en estos parámetros que en este caso determinan cuántos panes con y sin gluten existen en la cocina. Es decir, en la definición del nuevo problema se definen la lista de sucesión de valores numéricos mediante parámetros como `(next number_0 number_1)` o `(next number_1 number_2)` , lo que permite tanto sumar y restar una unidad como saber cuál es el menor valor posible que se puede definir (en este caso, el cero). Así, la comprobación que habría que hacer en este dominio para poder decrementar o incrementar un valor de un parámetro es si existe dicho sucesor o predecesor de dicho valor entre las definiciones del problema. Si no tienes predecesor de un número por ejemplo, sabes que el valor que tiene es cero.

Para que esto se entienda mejor se va a explicar el funcionamiento de uno de los operadores, por ejemplo de *make_sandwich_no_gluten*. Las precondiciones que tiene este operador son que existan los hechos `(at_kitchen_bread ?bread-gluten ?bread-no-gluten)` ,
`(at_kitchen_content ?content-gluten ?content-no-gluten)` y
`(at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)` , que exista el predecesor del pan y el contenido para saber que existe al menos una unidad de cada uno de los componentes que se van a utilizar `(next ?bread-no-gluten-minor ?bread-no-gluten)` y `(next ?content-no-gluten-minor ?content-no-gluten)` y que exista el sucesor del sándwich que se va a generar para que se pueda incrementar el número de sándwiches
`(next ?sandwich-no-gluten ?sandwich-no-gluten-major)` . El número máximo de sándwiches por tanto está definido por el mayor número definido en un parámetro `(next ?n ?maximo)` , siendo el valor de `?maximo` el número de cláusulas *notexist* del problema original. Finalmente, este operador elimina los hechos `(at_kitchen_bread ?bread-gluten ?bread-no-gluten)` ,
`(at_kitchen_content ?content-gluten ?content-no-gluten)` y
`(at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)` mediante la negación de los mismos en los efectos de este operador para así poder incrementar o decrementar sus valores `?bread-no-gluten` , `?content-no-gluten` y `?sandwich-no-gluten-major)`). Es decir, con los hechos definidos inicialmente el el problema `(next ?num ?numPlusOne)` se obtiene la escala de valores cuantitativos, pero al no ser esta definición numérica con dígitos, no es posible modificar

los hechos para incrementar o decrementar su valor si no es mediante el borrado del hecho inicial y la declaración de un nuevo hecho que contenga el nuevo valor.

Por último, esta solución es completa y óptima, puesto que es una redefinición de los operadores originales para adaptarlos a la lógica de un dominio numérico con proposiciones que sin embargo conserva por completo la lógica inicial del problema, por lo que a diferencia de otras versiones como en la 01 en que es necesario conservar los operadores originales para no perder la optimalidad, en este caso no es necesario, pues todas las acciones del dominio original se pueden realizar de manera atómica en este dominio (a diferencia de lo que ocurre con los dominios que usan macro-operadores, que es necesario aplicar varias acciones de las acciones agrupadas en el nuevo dominio para la aplicación de acciones individuales del dominio original). A continuación se incluye la definición de uno de los problemas.

6.9.2.5 Dominio 08

Esta versión sería la equivalente realizada a mano a la versión del dominio numérico con dígitos generado automáticamente. A diferencia de lo que ocurre con las versiones numéricas con palabras, en este caso la representación generada automáticamente y la representación manual son muy similares. De hecho, los nombres de los predicados utilizados en esta versión han sido cambiados para que coincidiesen con los obtenidos en la versión automática ya que de esta manera ambas versiones tienen una representación común de los problemas. Respecto a la representación del dominio, existen algunas variaciones tanto en relación con la versión numérica con palabras realizada a mano como con la versión automática numérica con dígitos.

En relación con la versión 05 del problema, el número de acciones, sus precondiciones y efectos son idénticos. Lo único que cambia entre ambas versiones es que la representación numérica de los predicados en este caso es con dígitos y en el anterior es con palabras.

Respecto al dominio numérico con dígitos generado automáticamente, lo único que varía entre ambos es que en el dominio automático se ha fragmentado en dos operadores el operador de este dominio `serve_sandwich_no_gluten` . Este operador sirve indistintamente un sándwich sin gluten a niños alérgicos al gluten y a niños no alérgicos al gluten, puesto que se ha eliminado de las precondiciones del operador si el niño es alérgico o no al gluten. Por otro lado, en el dominio generado automáticamente este operador ha sido dividido en otros dos al incluir en las precondiones si el niño es o no alérgico al gluten, teniendo por tanto un operador para

cada uno de estos dos supuestos.

6.9.2.6 Dominio 09

En esta versión se modifican tanto los problemas como el dominio. La modificación realizada consiste en eliminar el parámetro `(waiting ?c - child ?p - place)` para incluir la posición de los niños en `(allergic_gluten ?c - child ?p - place)` y `(not_allergic_gluten ?c - child ?p - place)` . La idea de esta modificación es que los parámetros `waiting` y `allergic_gluten` o `not_allergic_gluten` siempre aparecen juntos en los mismos operadores para referirse al mismo valor de `child` , por lo que con esta modificación se intenta comprobar si este tipo de uniones de parámetros cuyo uso coincide en los mismos operadores reduce o no el tiempo de búsqueda.

Los únicos operadores que ha sido necesario modificar son `serve_sandwich` y `serve_sandwich_no_gluten` , sustituyendo `(waiting ?c ?p)` y `(allergic_gluten ?c)` o `(not_allergic_gluten ?c)` por los nuevos parámetros `(allergic_gluten ?c - child ?p - place)` y `(not_allergic_gluten ?c - child ?p - place)` . A continuación se incluye un problema definido para esta versión.

Esta versión es completa y óptima puesto que sólo se ha cambiado la representación de los parámetros de algunos operadores pero no se ha modificado la funcionalidad original de los mismos.

6.9.2.7 Dominio 11

Esta versión requiere de la modificación de tanto los dominios como de los problemas, y la modificación consiste en eliminar el predicado `(not_allergic_gluten ?c - child)` para tratarlo en los operadores como la negación del predicado `(allergic_gluten ?c - child)` , es decir, como `(not(allergic_gluten ?c - child))` . Esto implica eliminar en los problemas los parámetros `(allergic_gluten ?c - child)` y modificar únicamente el operador `serve_sandwich` para definir al niño no alérgico al gluten que se le va a servir el sándwich de la manera que se ha decrito en la oración anterior. Un ejemplo de problema definido de esta forma sería el siguiente.

Esta versión es completa y óptima, puesto que es un cambio en la representación de la lógica, pero permanece la lógica de la versión original.

6.9.2.8 Dominio 12

Esta es con diferencia la versión con mayor conocimiento del dominio. Para la creación de este dominio se ha buscado el método resolutivo que permite obtener siempre la solución óptima para cualquier tipo de problema de entrada para después crear una versión de los dominios y problemas que se acercase lo máximo posible a esta solución. A continuación se encuentra el algoritmo de resolución óptimo para cualquier tipo de problema.

1. Contar el número de niños a los que hay que servir que son alérgicos al gluten y guardar ese valor en a.
2. Contar el número de niños a los que hay que servir que no son alérgicos al gluten y guardar ese valor en b.
3. Hacer a sándwiches sin gluten.
4. Hacer b sándwiches con o sin gluten.
5. Si no existe ninguna bandeja en la cocina, mover una bandeja a la cocina c.
6. Mover $(a + b)$ sándwiches a la misma bandeja c.
7. Para cada localización d en la que exista al menos un niño que tiene que ser servido:
   (a) Mover la bandeja c a d.
   (b) Para cada niño e que se encuentre en d y no haya sido servido:
      i. Servir el sándwich correspondiente (según sea o no alérgico al gluten) al niño e en d de la bandeja c.

En esta versión se ha conseguido replicar este algoritmo realizando las modificaciones en el dominio original que se expresan a continuación.

En cuanto a los **predicados**, el único predicado que se ha suprimido es `(notexist ?s - sandwich)` , ya que su función es limitar el número de sándwiches que se pueden producir y este concepto ya está incluído en los nuevos predicados `(num_sandwich_no_gluten ?n - num)` y `(num_sandwich_gluten ?n - num)` . Estos dos predicados son parámetros introducidos por el usuario con el número de sándwiches (definidos en numérico con proposiciones) con y sin gluten que es necesario hacer y que debe coincidir con el número de niños alérgicos o no al gluten que deben ser servidos. Ambos parámetros no sólo limitan el número de sándwiches que se pueden hacer, ya que en los operadores de creación de sándwiches se verifica que

el número de sándwiches restante de cada tipo sea mayor que cero, sino que también limitan los sándwiches con gluten que se pueden hacer y permiten separar los operadores en varias fases, impidiendo por ejemplo que se mueva una bandeja si antes no se han creado todos los sándwiches con y sin gluten que son necesarios. A su vez, se han incluido los predicados `(next ?minor - num ?major - num)` , que sirve para la creación de variables numéricas con proposiciones tal y como se especifica en el dominio 05, `(not_trays_kitchen)` es simplemente un operador creado para suplir la ausencia de comparaciones de tipo `forall` en *PDDL*, que permitiría saber si existen bandejas en la cocina y que en caso de que la posición inicial de ninguna de las bandejas esté en la cocina, será necesario añadir este predicado en el problema para indicar al dominio que es necesario ejecutar el operador `move_tray_kitchen` . El predicado `(first_load)` se incluye siempre en la definición del problema y sirve para poder saber cuándo se ha realizado la primera carga ya que así a partir de la segunda carga se va a exigir que la bandeja en la que se cargue el sándwich ya contenga al menos un sándwich (consiguiendo con ello que todos los sándwiches se carguen en la misma bandeja, ya que el operador para la primera carga `put_on_tray_first_load` que permite cargar el sándwich en cualquier bandeja sólo se va a poder ejecutar una única vez). Finalmente, el predicado `(num_sandwich_ontray ?n - num)` es inicializado con el número de sándwiches totales con y sin gluten que tienen que realizarse para así poder disminuir en una unidad cada sándwich cargado en la bandeja para que así se pueda poner como limitador de movimiento de las bandejas que ninguna se puede mover hasta que estén cargados todos los sándwiches.

Como en esta versión se han realizado importantes modificaciones en los **operadores**, por lo que van a ser explicados uno a uno como se realizó en el dominio original.

- **make_sandwich_no_gluten (porción de pan sin gluten, porción de contenido sin gluten, número de sándwiches actuales sin gluten, siguiente número de sándwiches sin gluten)**
  - **Precondiciones**: que exista al menos un contenido y un trozo de pan donde ambos sean sin gluten y exista un predecesor del número de sándwich sin gluten por hacer.
  - **Efectos**: eliminación de los dos componentes utilizados, decremento del número de sándwiches sin gluten que quedan por hacer y creación de un sandwich sin gluten.
- **make_sandwich (porción de pan, porción de contenido, número de**

sándwiches actuales con gluten, siguiente número de sándwiches con gluten)

  - **Precondiciones**: que exista al menos un contenido y un trozo de pan donde cualquiera o ambos de ellos pueden contener o no gluten, que exista un predecesor del número de sándwich con gluten por hacer y que se hayan hecho todos los sándwiches necesarios sin gluten.
  - **Efectos**: eliminación de los dos componentes utilizados, decremento del número de sándwiches con gluten que quedan por hacer y creación de un sandwich etiquetado como con gluten.

- **move_tray_kitchen (bandeja portátil, posición inicial, hecho auxiliar)**
  - **Precondiciones**: que exista el hecho que indica que no hay ninguna bandeja en la cocina y se hayan hecho todos los sándwiches con y sin gluten que había que hacer.
  - **Efectos**: mueve una bandeja a la cocina y eliminación del hecho que indica que no hay ninguna bandeja en la cocina para que así no se pueda repetir esta acción con más bandejas y así se pierda la optimalidad.

- **put_on_tray_first_load (número de sándwiches en la cocina, bandeja que se encuentra en la cocina, número de sándwiches en la bandeja anterior, hecho de que es la primera carga)**
  - **Precondiciones**: que se hayan hecho todos los sándwiches con y sin gluten que había que hacer, que haya al menos una bandeja en la cocina y que sea la primera carga en una bandeja.
  - **Efectos**: carga en una bandeja un sándwich.

- **put_on_tray (número de sándwiches en la cocina, bandeja que se encuentra en la cocina, número de sándwiches en la bandeja anterior, hecho de que no es la primera carga)**
  - **Precondiciones**: que se hayan hecho todos los sándwiches con y sin gluten que había que hacer, que haya al menos una bandeja en la cocina y que la bandeja sobre la que se va a cargar tenga ya al menos un sándwich.
  - **Efectos**: carga un sándwich en una bandeja previamente cargada.

- **move_tray (bandeja, posición inicial, hecho de que se han cargado todos los sándwiches, posición niño en espera no alérgico al gluten)**

  - **Precondiciones**: que se hayan hecho todos los sándwiches que había que hacer con y sin gluten, que se hayan cargado todos los sándwiches en la bandeja y que en la posición actual no quede ningún niño al que se le pueda servir un sándwich con gluten.
  - **Efectos**: mueve la bandeja a una nueva posición.

- **move_tray_no_gluten (bandeja, posición inicial, hecho de que se han cargado todos los sándwiches, posición niño en espera alérgico al gluten)**
  - **Precondiciones**: que se hayan hecho todos los sándwiches que había que hacer con y sin gluten, que se hayan cargado todos los sándwiches en la bandeja y que en la posición actual no quede ningún niño al que se le pueda servir un sándwich sin gluten.
  - **Efectos**: mueve la bandeja a una nueva posición.

- **serve_sandwich_no_gluten (niño no alérgico al gluten, niño esperando, posición niño, posición bandeja)**
  - **Precondiciones**: que al menos una de las bandejas portátiles esté en la misma posición que uno de los niños, que la bandeja que cumpla la precondición anterior tenga al menos un sándwich sin gluten, que un niño no alérgico al gluten esté en la misma localización que la bandeja que cumple las precondiciones anteriores y que este mismo niño esté esperando un sándwich.
  - **Efectos**: el niño pasa a estar servido y se elimina el sándwich utilizado en la operación.

- **serve_sandwich (niño alérgico al gluten, niño esperando, posición niño, posición bandeja)**
  - **Precondiciones**: que al menos una de las bandejas portátiles esté en la misma posición que uno de los niños, que la bandeja que cumpla la precondición anterior tenga al menos un sándwich con o sin gluten, que un niño no alérgico al gluten esté en la misma localización que la bandeja que cumple las precondiciones anteriores y que este mismo niño esté esperando un sándwich.
  - **Efectos**: el niño pasa a estar servido y se elimina el sándwich utilizado en la operación.

6.10 Versiones generadas automáticamente

En esta sección se explicará brevemente la composición de las versiones generados automáticamente mediante software independiente de dominio de terceros. Para todos estos dominios se ha utilizado exclusivamente como entrada el problema y el dominio original.

Como la lógica de estas modificaciones es de otros propietarios, las explicaciones serán bastante menos detalladas que las que aparecen en la sección de los dominios modificados a mano.

6.10.1 Versión PTT

Este modificador debería transformar el dominio mediante la inclusión de macrooperadores, que como se ha mostrado en la sección de dominios hechos a mano, existen varias alternativas para dicha implementación. Sin embargo, el software de *PTT* ha sido incapaz de encontrar un solo macrooperador, por lo que tanto la definición del dominio como los problemas generados son los mismos que los introducidos como entrada, por lo que en lo referente a la comparación de los tiempos obtenidos con las distintas versiones esta versión de los problemas va a ser descartada, al ser exactamente igual que la versión original del problema.[4]

6.10.2 Versión numérica con proposiciones

Esta versión ha sido generada automáticamente mediante el software *Baggy*, que transforma los dominios en representaciones numéricas con proposiciones. Baggy añade una gran cantidad de predicados que hacen difícil de seguir su lógica para explicar en detalle su motivación, como es el caso cuando tiene por entrada el dominio y los problemas originales. Respecto a las acciones, estas incororan estos nuevos predicados tanto en las precondiciones como en los efectos así como también sucede tanto en la inicialización como en los efectos de los problemas.

Por último, al ser únicamente un cambio de representación, los operadores iniciales han sido conservados no así los predicados que estos contenían.

6.10.3 Versión numérica con dígitos

Esta versión modifica la representación del dominio para que sea numérica con dígitos. Respecto a la versión modificada a mano que es también numérica con dígitos, la representación de los problemas es la misma puesto que se han cambiado

[4]En ejecuciones de este software con otras versiones de este dominio, se ha comprobado que este software es capaz de encontrar macrooperadores. Sin embargo, al centrar el estudio en la comparación de las distintas versiones que tienen como punto de partida el dominio original, estas modificaciones han sido descartadas.

el nombre de los predicados de la versión manual para que así sea. En lo que atañe al dominio, todos los operadores son idénticos entre sí entre ambas versiones a excepción del operador para servir sándwiches sin gluten. Mientras que en la versión manual se ha quitado de las precondiciones si el niño que va a recibir el sándwich es alérgico o no para que así este operador sea válido para los dos casos, en el caso de la versión generada automáticamente este operador se ha desglosado en dos incluyendo en cada uno la precondición de si el niño que va a recibir el sándwich es alérgico o no al gluten.

Este desdoblamieto de operadores respecto al dominio generado a mano no aporta ningún beneficio, ya que el hecho de que el niño sea o no alérgico al gluten se encuentra en ambos casos exclusivamente en las precondiciones y no en los efectos. Es decir, se está incrementando respecto a la versión hecha a mano el número de precondiciones de un operador así como se está añadiendo un nuevo operador innecesario, siendo el resto de operadores idénticos entre ambas versiones (y que por ello no van a volver a ser explicados). Esta modificación por tanto incrementa el factor de ramificación sin ninguna contrapartida, por lo que por lo menos el tiempo de búsqueda debería ser igual o mayor que la versión manual. Esta hipótesis será comprobada experimentalmente en la sección de evaluación de resultados.

7 Resultados obtenidos

En esta sección se van a detallar las pruebas realizadas así como los resultados obtenidos con su correspondiente interpretación. La tendencia de estos resultados debería ser la misma independientemente de las características del hardware y del sistema operativo utilizado, sin embargo los resultados absolutos varían según estas dos características, motivo por el cual se incluyen estos datos.

Todas las pruebas a las que se va a mentar en esta sección son pruebas de eficiencia de los distintos planificadores con las distintas versiones de los problemas. No se ha realizado ninguna prueba funcional para comprobar la correctitud de los scripts ya que resulta más fiable y requiere menos tiempo verificar que el resultado de alguna prueba aleatoria de su ejecución coincide con la ejecución manual que diseñar pruebas funcionales que verifiquen lo que ya se ha hecho a mano.

7.1 Especificaciones del ordenador utilizado

Como ya se ha comentado, el hardware y el sistema operativo utilizado influyen en los resultados obtenidos en las pruebas de eficiencia realizadas. Por este motivo, se incluye a continuación estas características.

- Tipo de dispositivo: ordenador de sobremesa
- Modelo: Hewlett Packard 550-244NS A[5]
- Procesador: AMD A10 8750
- Frecuencia: 3.6GHz
- Memoria RAM: 16GB
- Sistema operativo: GNU-Linux Debian 8.7

7.1.1 Planificadores utilizados

Los planificadores utilizados para las pruebas realizadas son *Metric-FF* en su versión 2.1 y *LPG* en su versión *td*. Inicialmente también se incluyó el planificador FastDownward utilizando Lama2011 como heurística, pero se descartó este planificador al resolver menos del 10% del total de problemas para todas las versiones por debajo de los 30 minutos de tiempo límite de búsqueda del planificador por cada problema, lo que impide poder extraer conclusiones sólidas de la mejora entre las distintas representaciones de los problemas. Por otro lado, con los dos planificadores

[5] https://support.hp.com/es-es/document/c05045080

utilizados para estas pruebas el 100% de los problemas de alguna de esas versiones eran resueltos en su totalidad por ambos planificadores.. El funcionamiento interno de estos tres planificadores queda descrito en la sección del estado del arte.

Podría haberse aumentado el número de planificadores utilizados para la sección de pruebas. Sin embargo, teniendo en cuenta que una estimación muy benévola de la sección de planificación del proyecto se estimaba el número de horas de ejecución de las pruebas funcionales en 139 horas de ejecución, la inclusión de pruebas con nuevos planificadores implicarían un gasto de tiempo considerable. Si se hubiese dispuesto de más tiempo, se podrían haber incluído pruebas con más planificadores. Estas pruebas posteriores se cree que podrían haber servido para entender la relación de las modificaciones realizadas con un entorno más amplio de planificadores para así quizá poder sacar conclusiones apriorísticas sobre el tipo de modificaciones que pueden beneficiar a un tipo concreto de planificador. Esto por falta de tiempo no ha podido ser realizado por lo que se encuentra entre las líneas de trabajo por las que se podría continuar este proyecto.

Existe también otra alternativa que es realizar las pruebas exclusivamente con un único planificador. Esta opción podría hacer que se extrajesen conclusiones equivocadas o muy limitadas respecto a las modificaciones realizadas en los dominios, puesto que la mejora en la planificación que se puede obtener con dichas modificaciones depende del planificador y de la heurística que éste utilice.

Los parámetros de ejecución utilizados para estos planificadores son:

- **Metric-FF**: -s 0 (*FF* estándar: *EHC+H* y *BFS*).
- **LPG-td**: -n 1 (genera únicamente una solución).

Respecto al coste de las acciones, la ejecución de las acciones de todas las versiones tienen coste 1 salvo en el caso de los dominios con macrooperadores, en cuyo caso los macrooperadores tienen el coste del número de operadores del dominio original que estén sustituyendo.

7.1.2 Métricas utilizadas

Los problemas han sido ejecutados por cada planificador con un tiempo máximo de 30 minutos por cada problema, que es el tiempo límite utilizado en las competiciones de la *IPC*. En caso de no encontrar una solución en ese lapso de tiempo, el problema contabiliza como no resuelto.

Las métricas utilizadas para la comparación de los resultados han sido estas tres:

- Tiempo necesario para encontrar la solución.
- Longitud del plan (en caso de encontrar solución).
- Calidad relativa de la solución según la métrica de la *IPC*.

7.2 Pruebas realizadas

Las pruebas que se han realizado tenían como fin demostrar una hipótesis surgida de una mezcla de intuición y de la revisión del estado del arte: que tanto el tiempo de búsqueda de un planificador como la calidad de la solución de un mismo problema podía variar con la modificación de la representación del dominio y/o del problema. El software utilizado de otros investigadores que ha sido utilizado para estas pruebas, como se explicó en la sección de la solución propuesta, tiene sus limitaciones y no es capaz de encontrar siempre las mejores modificaciones que por otra parte sí son posibles encontrar a mano. Esto es algo que intuitivamente era de esperar. En esta sección se intentará cuantificar el coste que supone la automatización de estas modificaciones.

7.3 Resultados obtenidos

En las siguientes secciones, va a a intentar medir la calidad de cada una de las versiones para cada uno de los dos planificadores analizados en relación a estas tres métricas mencionadas en el apartado anterior.

Para evitar saturar las gráficas con demasiadas funcione que impidan ver con claridad los resultados obtenidos, se han separado las distintas versiones a comparar en grupos de 1 a 4 versiones según la similitud de los cambios realizados. A su vez, se ha incluido en todos ellos la representación de la función de la versión original para poder ver el grado de mejora o pérdida respecto a ésta.

7.3.1 Comparación de las versiones 01 y 03

En esta sección, se van a comparar los resultados obtenidos al ejecutar las siguientes versiones de los problemas que se han generado utilizando macrooperadores (es decir, la versión 01 y 03 de los problemas realizados a mano).

Como ya se comentó en la sección de la solución propuesta, en esta sección no se van a incluir los tiempos de ejecución de la versión obtenida mediante las modificaciones de PTT puesto que este software no fue capaz de realizar una sola modificación del

dominio original, por lo que todos los resultados funcionales serían idénticos a los del dominio original puesto que es el mismo.

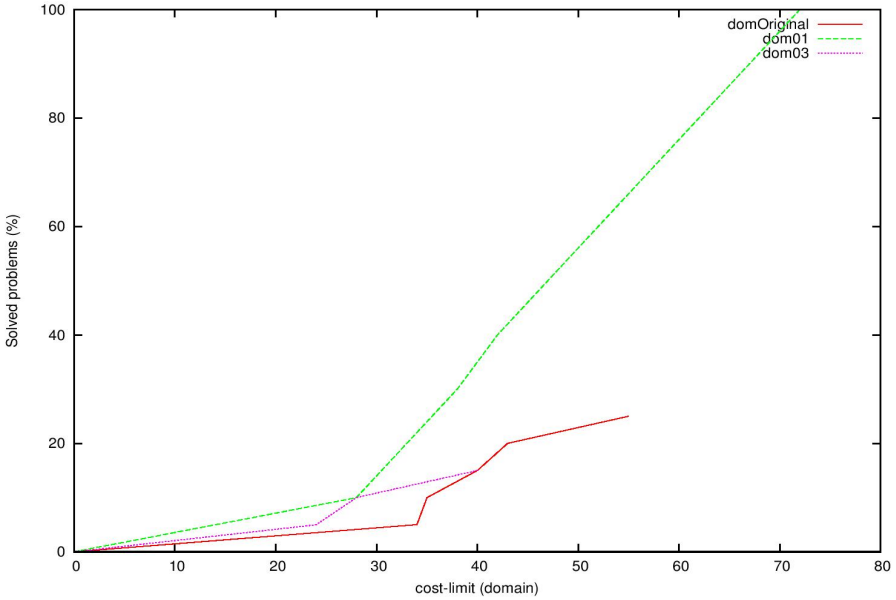

Figura 4: MFF-original-01-03-coste

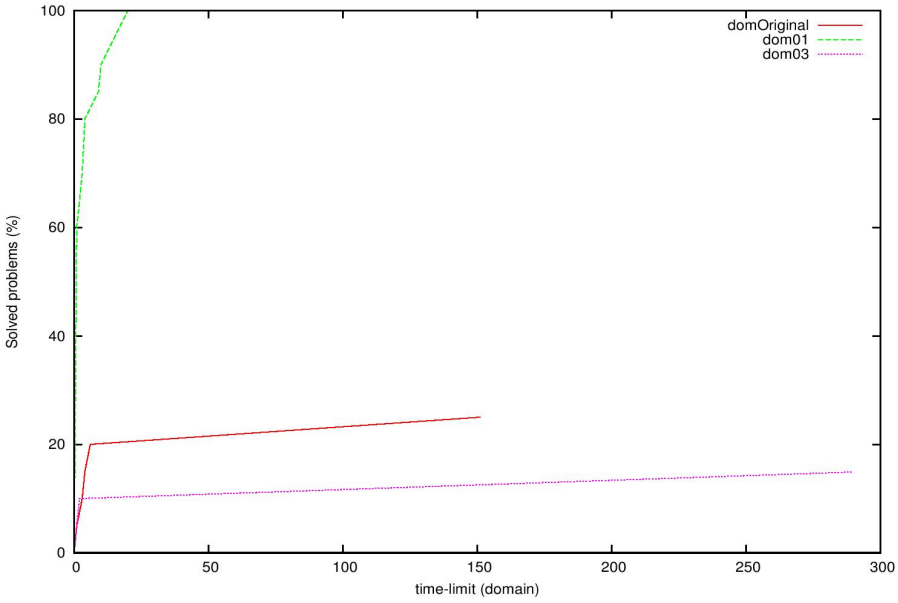

Figura 5: MFF-original-01-03-tiempo

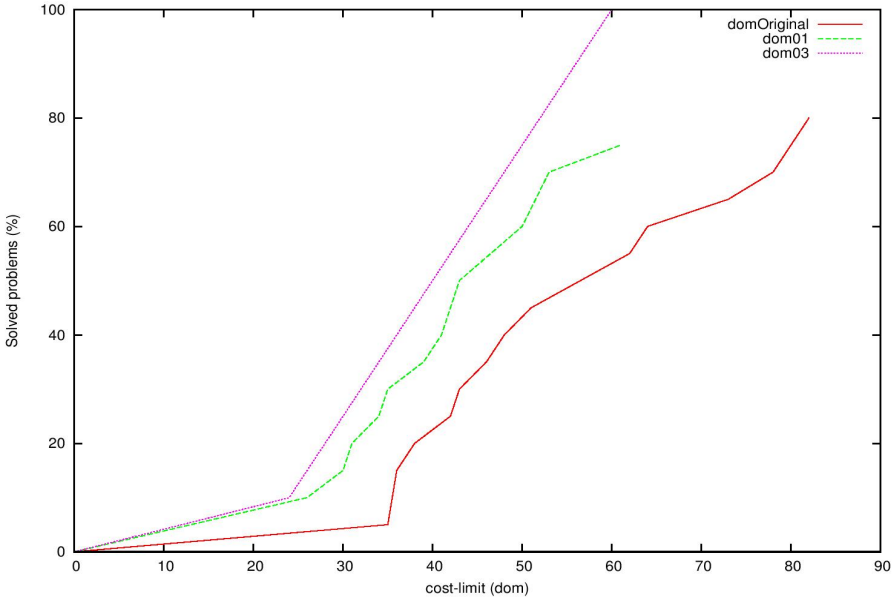

Figura 6: LPG-original-01-03-coste

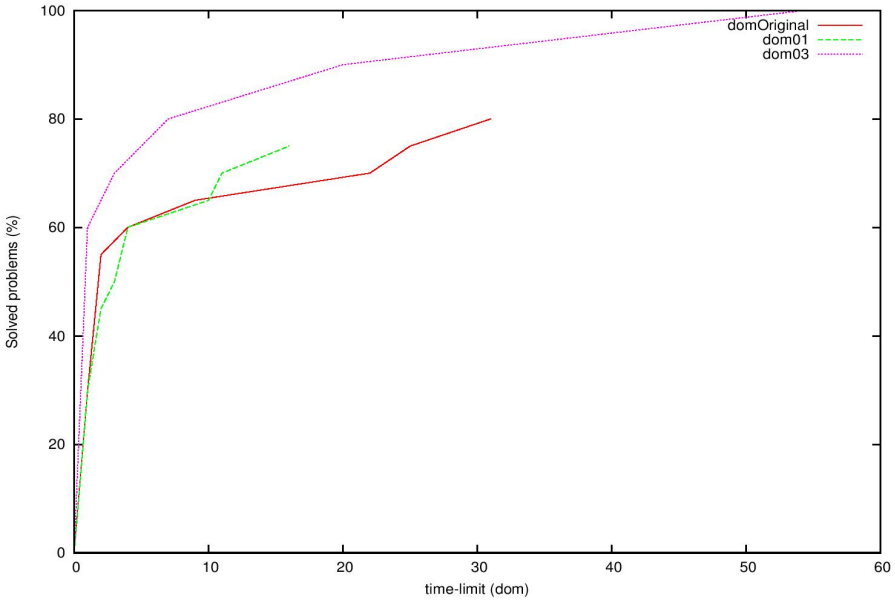

Figura 7: LPG-original-01-03-tiempo

En los resultados obtenidos con *MFF* se puede ver en la figura 4 que el coste para resolver los mismos problemas es mayor en la versión 01 que tanto en la versión original como en la versión 03. De hecho, la que tiene un menor coste de las tres es la versión original. Por otro lado, en la versión 01 consigue resolver todos los problemas, a diferencia de lo que ocurre en la versión 03 que resuelve además menos problemas que la versión original. A esto hay que añadir que el tiempo de ejecución para la resolución de los problemas es muy inferior en el caso de la versión 01 y es superior en el caso de la versión 03, tal y como se observa en la figura 5.

La interpretación que se hace de estos resultados es que mientras que en el primero la *utilidad* del macrooperador está ajustada para que contrarreste los efectos negativos que produce por ejemplo un mayor coste para obtener la solución, este no es el caso de la versión 03. La versión 03 se ha hecho explícitamente para mostrar el resultado que se obtiene cuando se genera un macrooperador con un número de operadores demasiado grande que hace que la utilidad de dicho macrooperador sea negativa, como se puede ver en los resultados. Por otro lado, en la versión 01 se obtiene la ventaja de que al unir dos operadores que frecuentemente se suelen suceder (el operador de movimiento del carrito y el operador de de servir el sándwich), como se han retirado los operadores iniciales, esto reduce el factor de ramificación del árbol de búsqueda pero a su vez implica obtener soluciones más costosas puesto que si por ejemplo dos niños están en la misma posición, en lugar de conseguir la solución óptima que sería cargar los dos sándwiches en una bandeja (2), mover la bandeja a la localización de los niños (1) y servirles (2), lo que supone un gasto de 5 unidades, con esta solución habría que cargar un sándwich en la bandeja (1), mover el carro y servir a un niño (2), mover el carro de vuelta a la cocina (1), cargar un sándwich (1) en la bandeja, mover la bandeja y servir el sándwich (2), lo que supone un gasto de 7 unidades, dos más que en la solución original.

Respecto a los resultados obtenidos con *LPG-td* que se muestran en las figuras 6 y 7, tal y como se puede observar en la figura 7, con la versión 03 se consiguen resolver todos los problemas en el menor de los tiempos posibles de las tres versiones. En este caso, la versión 01 resuelve menos problemas que en la versión original pero los problemas que resuelve lo hace en menos tiempo que en la versión original.

7.3.2 Comparación del dominio02

En esta sección, se va a comparar la pequeña modificación realizada en uno de los operadores del dominio original respecto a la versión original. La modificación únicamente deja de etiquetar como que tuviesen gluten los sándwiches generados con dos componentes sin gluten que aparezcan como entrada del operador `make_sandwich` . Este es un caso muy concreto en el que como se comentó en la sección de la solución propuesta, debería reducir el tiempo de búsqueda para encontrar la solución pero muy ligeramente, puesto que como se ha dicho no es un caso genérico, sino uno muy concreto para una única operación.

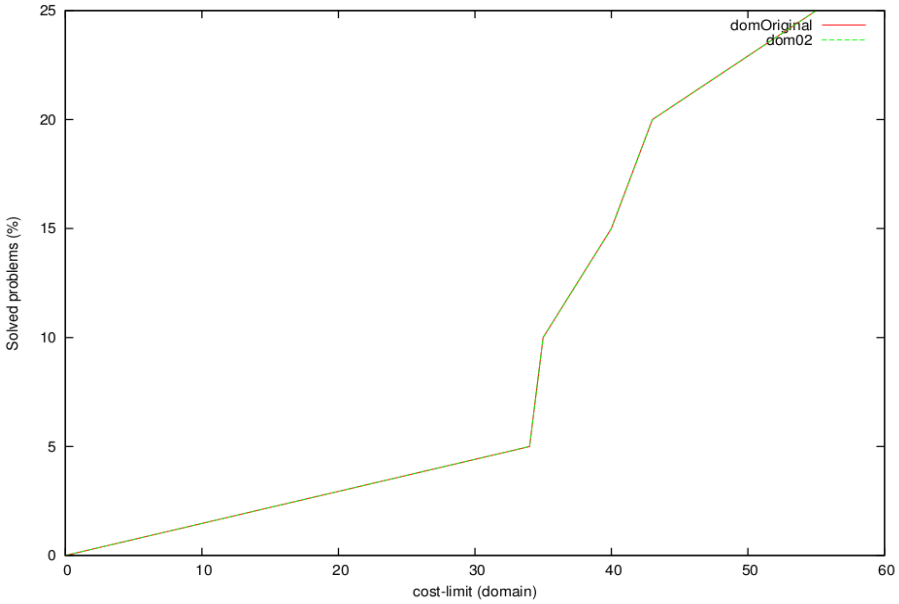

Figura 8: MFF-original-02-coste

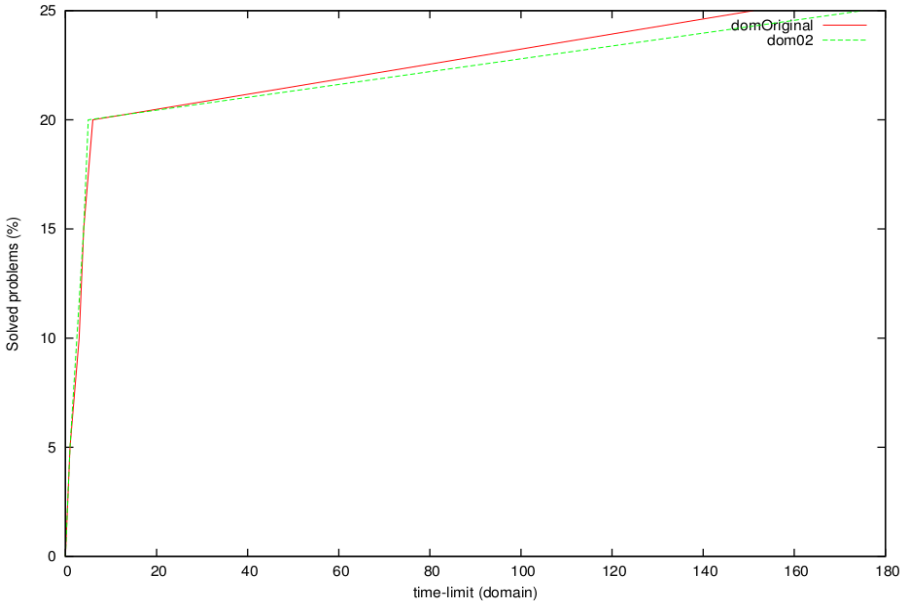

Figura 9: MFF-original-02-tiempo

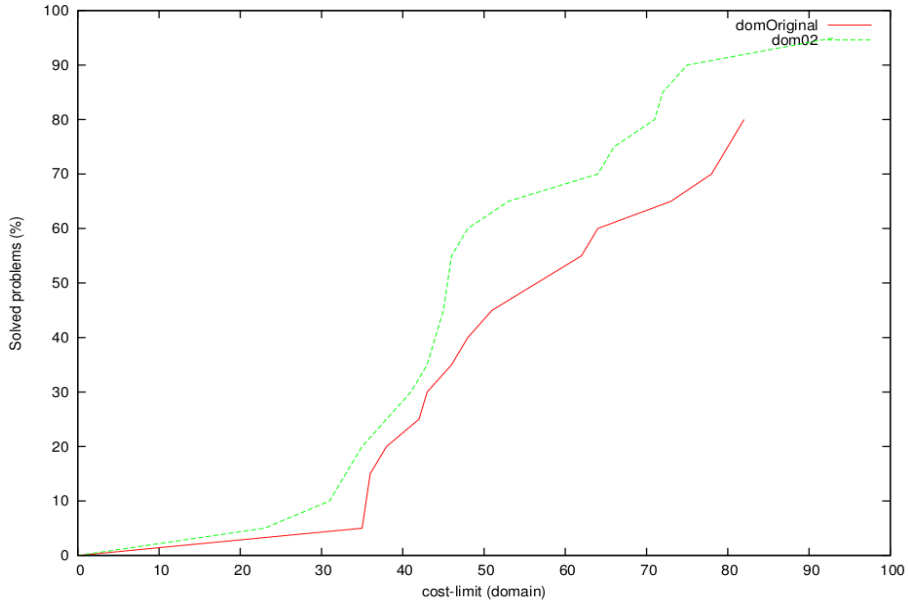

Figura 10: LPG-original-02-coste

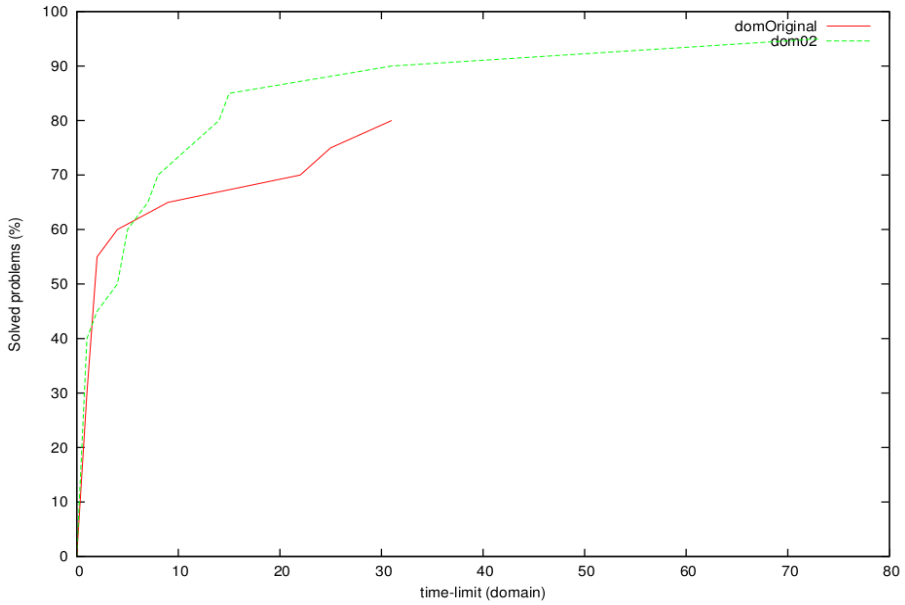

Figura 11: LPG-original-02-tiempo

Como se puede ver en las figuras 8 y 9, esta versión reduce ligeramente el tiempo de búsqueda y esta mejora se va acrecentando a medida que aumenta el tamaño de los problemas. Esto es debido a que con el aumento del tamaño de los mismos, aumenta la probabilidad de que ocurra el caso particular en el que el segundo dominio provoca una reducción del tiempo de búsqueda. Respecto al coste, se puede ver en la figura 9 que es idéntico en ambos casos, puesto que realmente lo que se consigue con esta nueva versión es podar del árbol de búsqueda soluciones duplicadas (que podrían ser obtenidas mediante el otro operador de crear sándwiches con gluten), por lo que la solución final encontrada por ambas versiones es muy probable que acabe siendo la

misma.

Las figuras 10 y 11 obtenidas con LPG-td son más difíciles de interpretar y no siguen el mismo patrón que en *Metric-FF*. En estas, con la nueva versión sí se producen importantes modificaciones tanto en el tiempo consumido como en el coste de las soluciones obtenidas. Respecto al tiempo consumido, con la versión original se resuelven 3 problemas menos y los problemas resueltos requieren de más tiempo. Por otro lado, se puede ver en la figura 10 que el coste de resolución de estos problemas es también superior en la versión original.

7.3.3 Comparación del dominio09 y dominio11

En estas versiones, se realiza exclusivamente la modificación de los predicados aumentando o disminuyendo su tamaño. En el caso de la versión 09, se aumenta dicho tamaño al juntar los predicados de `allergic_gluten` y `not_allergic_gluten` con el predicado `waiting` , ya que en ambas partes de los predicados originales se refiere a la misma variable `child` . En la versión 11, se sustituye el predicado de negación `not_allergic_gluten` por la negación del predicado `allergic_gluten` es decir, `(not(allergic_gluten))` .

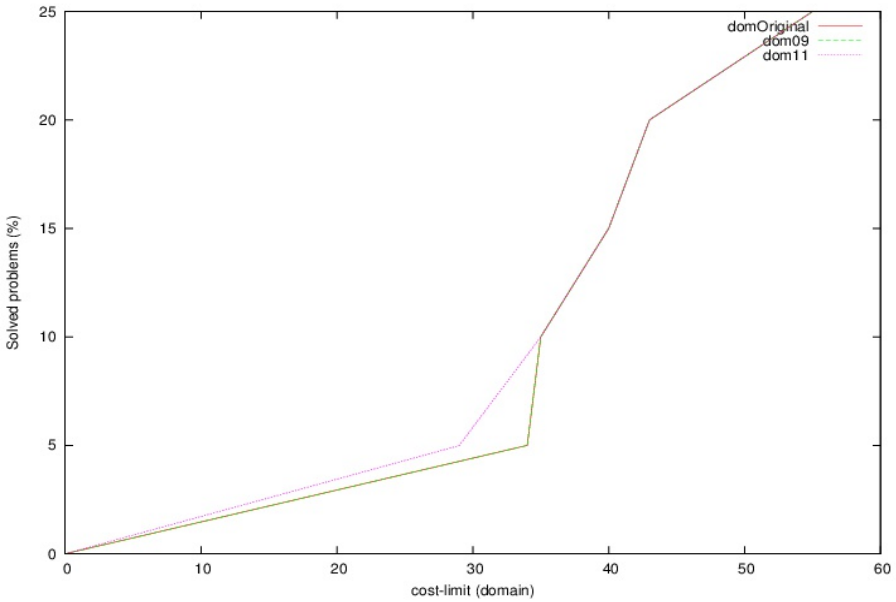

Figura 12: MFF-original-09-11-coste

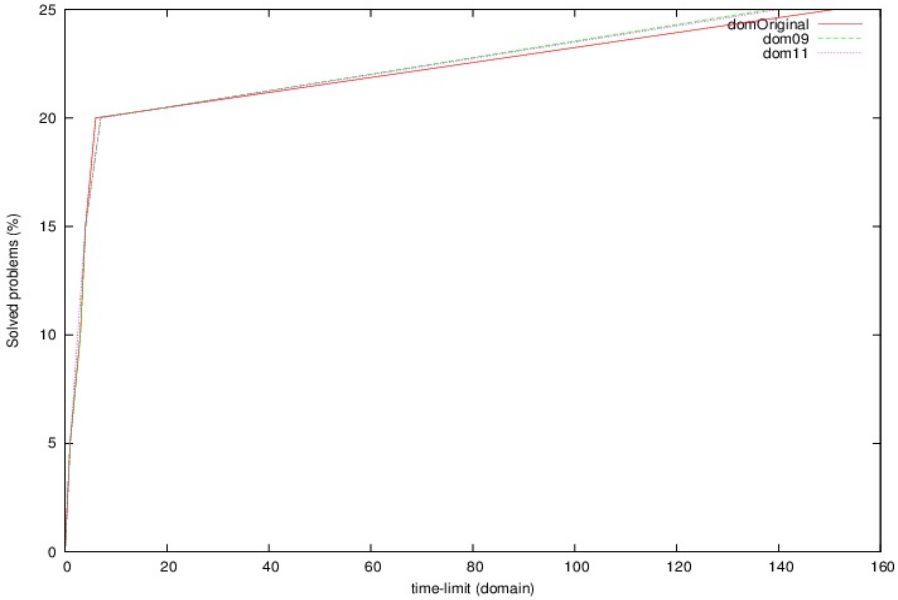

Figura 13: MFF-original-09-11-tiempo

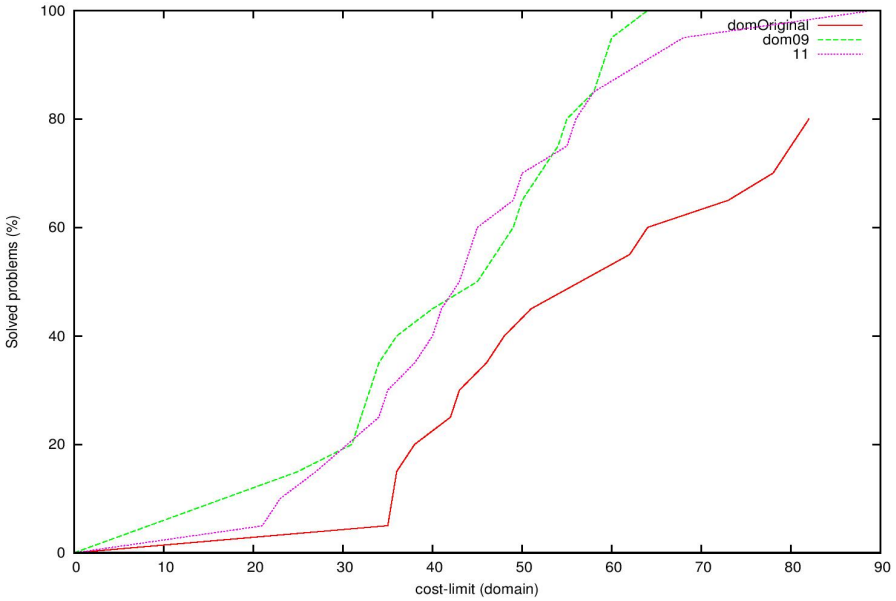

Figura 14: LPG-original-09-11-coste

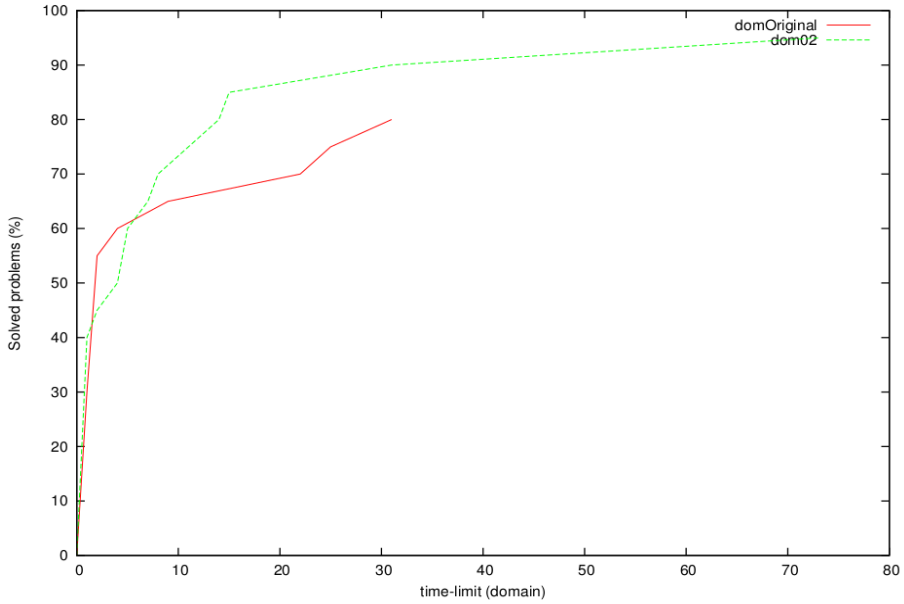

Figura 15: LPG-original-09-11-tiempo

Respecto a los resultados obtenidos con MFF que se muestran en las figuras 12 y 13, con la versión 09 se obtienen soluciones con exactamente el mismo coste que en la versión original a diferencia de la versión 11, con la que se obtienen problemas con menor coste hasta llegar a problemas con una complejidad en la que convergen los costes de las tres versiones. Respecto a los problemas y el tiempo de resolución, todos convergen en la resolución de el 25% de los problemas pero con una ligera reducción de tiempo de búsqueda en las dos versiones realizadas a mano, tal y como se puede observar en la figura 13. Al haberse realizado modificaciones pequeñas de los predicados, no se esperaban grandes cambios en la calidad y los tiempos de ejecución, por lo que era previsible que estos resultados fueran cercanos a los obtenidos con la versión original.

En relación a los resultados obtenidos con LPG-td que se muestran en las figuras 14 y 15, estos son mucho más impredecibles y difíciles de interpretar. Ambas versiones consiguen aumentar el porcentaje de problemas resueltos de en torno al 80% al 100%. Se puede apreciar en la figura 14 que el coste de dichas resoluciones es inferior que en el caso original y dependiendo del tramo es mayor o menor el coste entre ambas versiones. Finalmente, respecto al tiempo de ejecución, se puede apreciar en la figura 15 que los menores tiempos se obtienen con la versión 09, aunque con la versión 11 también se reducen los tiempos de ejecución respecto a la versión original.

7.3.4 Comparación de los dominios numéricos

En esta sección, se van a comparar tanto los dominios numéricos realizados a mano como los dominios numéricos obtenidos obtenidos mediante software independiente

de dominio de otros desarrolladores. Respecto a los dominios numéricos realizados a mano, la versión 05 corresponde a la versión numérica con proposiciones mientras que la versión 08 corresponde a la versión numérica con dígitos.

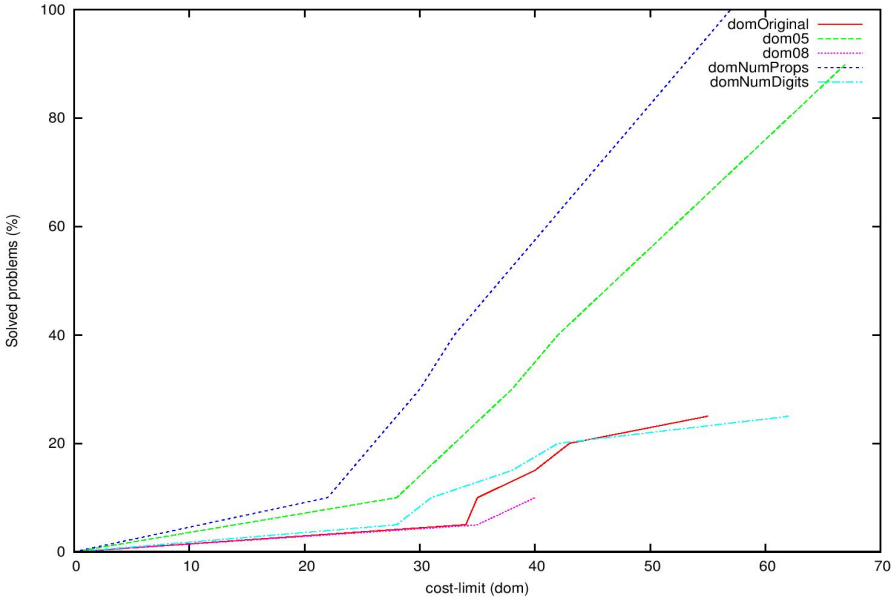

Figura 16: MFF-original-02-coste

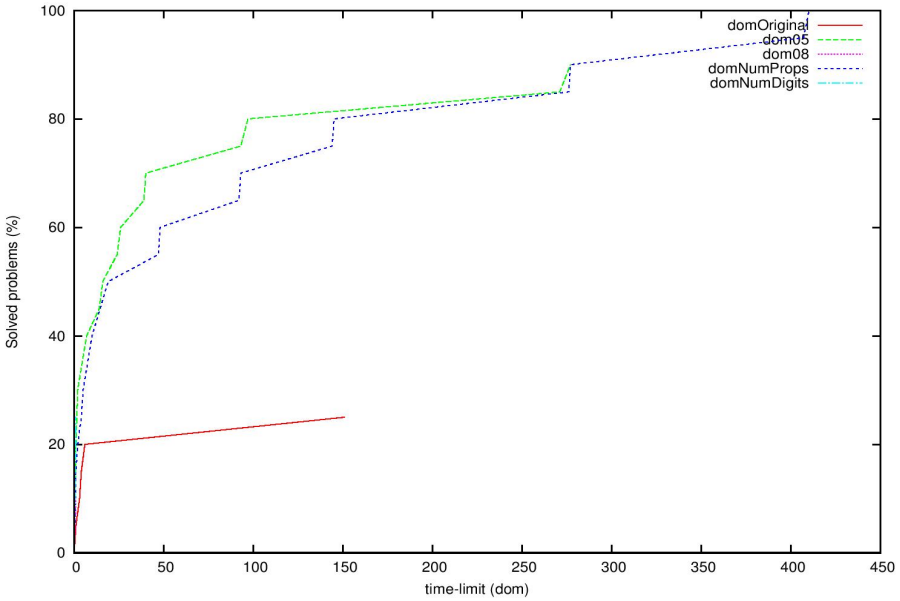

Figura 17: MFF-original-02-tiempo

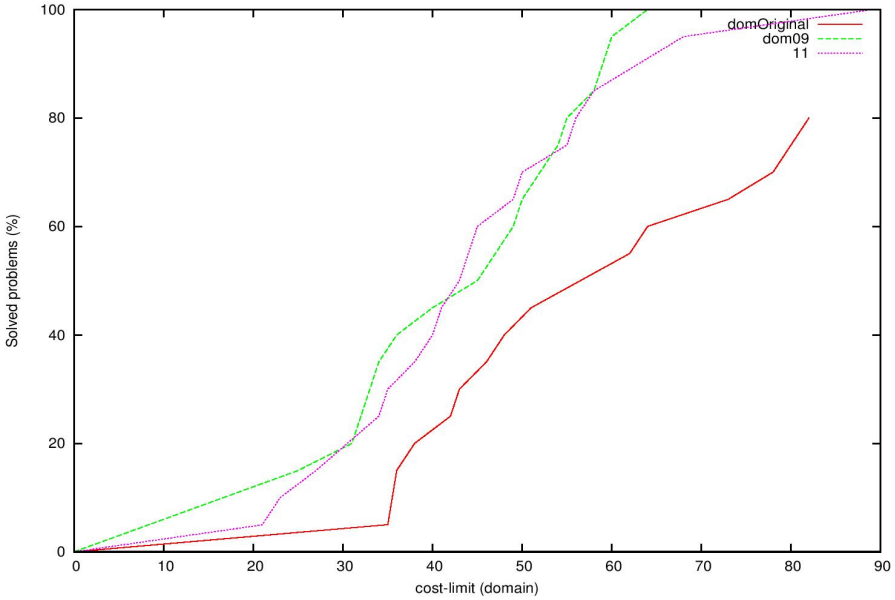

Figura 18: LPG-original-02-coste

Figura 19: LPG-original-02-tiempo

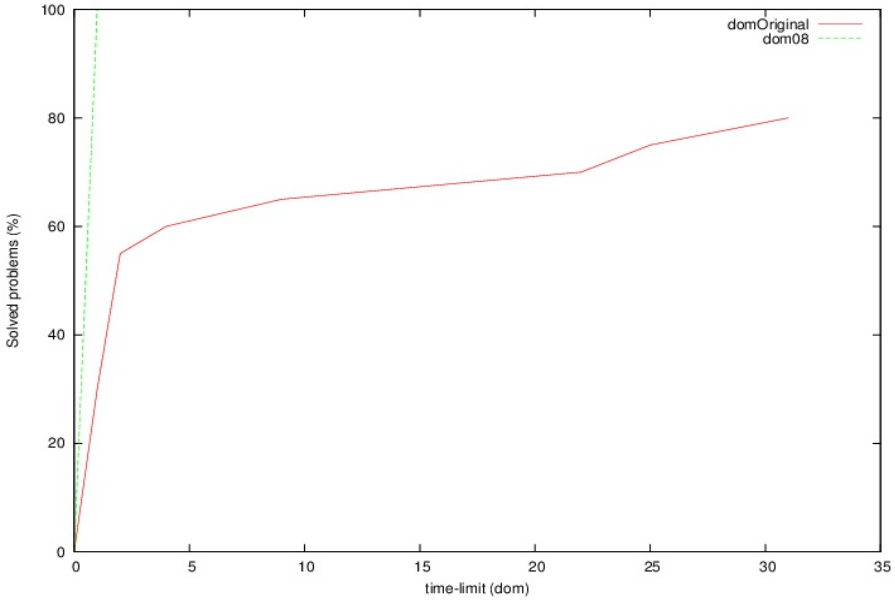

Figura 20: LPG-original-08-tiempo

En la figura 17, prácticamente no se ve el resultado de la versión numérica con dígitos generada automáticamente. Esto se debe a que está superpuesta con el principio de la versión 05. Al quedarse la gráfica en un valor de resolución de problemas del 25% de los problemas de entrada, queda patente que esta versión comparada con el resto de versiones no es la mejor de las soluciones. Sin embargo, este resultado coincide con el porcentaje de problemas resueltos por la versión original requiriendo por otro lado mucho menos tiempo. Por otro lado, supera en el número de problemas

resueltos y en un coste similar a la versión 08, tal y como se ve en la figura 16. A su vez, el coste de las soluciones obtenidas con la versión numérica con dígitos es similar al coste de la versión original, variando la versión con mayor coste entre estas dos versiones entre los distintos problemas resueltos.

Los mejores resultados obtenidos con el planificador MFF se obtienen con las versiones numérica con palabras y la versión 05, resolviendo más problemas y con menor coste con la versión automática numérica con palabras, pero requiriendo más tiempo que en la versión 05.

Respecto a la versión numérica con proposiciones generada automáticamente y la versión realizada manualmente (versión 05), se puede observar en los resultados de las figuras 16 y 17 que con el planificador *MFF* que en términos generales la versión generada automáticamente obtiene mejores resultados. En este caso, es difícil intentar desgranar los motivos subyacentes puesto que la versión automática genera bastantes nuevos predicados que son complicados de interpretar y la versión final que se genera tiene una estructura totalmente distinta de la que aparece en la versión manual.

Por otro lado, en las versiones numéricas con dígitos, los resultados son muy similares puesto que los problemas de entrada tienen una representación común y en lo único que varían los dominios es en uno de los operadores (toda la información referente a este cambio se puede encontrar en la sección de solución propuesta). En los resultados, se puede ver en las figuras 16 y 17 que con la versión automática se resuelven más problemas en la versión automática, teniendo esta un coste mayor pero obteniendo estas soluciones en un tiempo similar.

Respecto a los resultados de *LPG-td* que aparecen en las figuras 18, 19 y 20, la versión automática numérica con proposiciones vuelve a resolver la totalidad de los problemas, al igual que también lo hace la versión numérica con proposiciones generada manualmente (la versión 08). Esto quiere decir que sin lugar a dudas la transformación automática para generar un dominio con proposiciones es beneficiosa para ambos planificadores.

Respecto a los costes para resolver los problemas, se puede ver en la figura 18 que los menores costes se obtienen con la versión 08, que en su tramo de problemas iniciales se superpone con el coste de la versión original. En lo que concierne a los tiempos, se puede ver en la figura 19 que la versión más rápida en la gráfica en la

que aparecen representados los tiempos de las 5 versiones pudiera parecer que es la versión original. Sin embargo, en realidad la versión más rápida es la versión 08, como se puede ver en la figura 20 que se ha añadido incluyendo exclusivamente las gráficas de tiempo de la versión que aparenta ser la más rápida (la original) y la que realmente es significativamente más rápida (la versión 08). En la figura 19 no se puede ver nítidamente el tiempo de la versión 08 debido a que como la diferencia de tiempos es tan grande respecto al resto de versiones, este tiempo está tan próximo a cero que es muy complicado ver estos tiempos en esta gráfica.

7.3.5 Comparación del dominio12

Este es un dominio por etapas en el que se ha intentado obtener la mejor versión posible respecto a tanto el tiempo de búsqueda como al coste de obtención de las soluciones. Para crear este dominio, han sido necesarios añadir nuevos operadores, que como se comentará a continuación, han aumentado el coste de las soluciones.

Como este es el dominio que ha sido diseñado específicamente para ser la mejor versión de todas, no solo se va a comparar el resultado con la versión original, sino que se va a comparar el coste de la solución de los problemas con el coste óptimo de la solución de cada uno de los problemas.

La solución óptima para cualquier problema de *Childsnack* (cuya lógica es en la que se ha basado esta versión del dominio) consiste en aprovechar que las bandejas tienen una capacidad infinita para únicamente utilizar una bandeja para la resolución del problema, puesto que utilizar más bandejas implica realizar al menos el mismo número de desplazamientos que con una única bandeja, por lo que para simplificar el árbol de búsqueda se va a acotar las soluciones obtenidas a aquellas en las que únicamente se hace uso de una bandeja. A su vez, para cada niño que haya que servir hará falta: realizar un sándwich, cargarlo en la bandeja y servirlo. Considerando que la bandeja está en la cocina y todos los niños están en la misma localización, como mínimo será necesario el desplazamiento desde la cocina a la localización en la que estén todos los niños que haya que servir (salvo en el caso de que todos los niños estén en la cocina, en cuyo caso no hace falta desplazamiento). Por otra, parte, en el peor de los casos, ninguna de las bandejas estará colocada inicialmente en la cocina (lo que supondrá el coste de una acción para desplazarla a la cocina para poder cargar los śandwiches) y cada niño estará en una posición distinta, por lo que a las acciones anteriores será necesario añadir una acción de desplazamiento de la bandeja por cada niño a servir. Por tanto, el número de pasos óptimo o coste para resolver cualquien problema de Childsnack queda definido por la siguiente fórmula:

$niños \times 3 \leq Costeoptimo \leq niños \times 4 + 1$

A partir de esta fórmula genérica, se puede acotar hasta saber directamente el coste exacto de la solución óptima si se cuenta con la información referente al número de localizaciones distintas en las que están los niños sin contar aquellos que puedan estar en la cocina (a) y si existe alguna bandeja en la cocina (b). Así, el coste exacto de la solución óptima para cualquier problema de Childsnack en su versión original es:

$Costeoptimo = niños \times 3 + a + b$

Con esta fórmula se obtienen los resultados de los costes óptimos para cada problema de entrada que se encuentra en la tabla 2. Por poner un ejemplo de cómo se obtienen este valor así como un plan que demuestre que existe una solución con el coste óptimo obtenido con esta fórmula, se va a obtener el coste óptimo y un plan óptimo para el problema 01-2 de la versión original.

En el problema 01-2, los datos que observamos que son necesarios meter como entrada para la obtención del coste óptimo son los siguientes:

- niños = 6
- a = 3 (los niños que hace falta servir están en las localizaciones `table1` , `table2` y *table3*.
- b = 0 (las bandejas portátiles `tray1` y `tray2` están en la cocina).

Por tanto, se obtendría que el coste óptimo para el problema 01-2 de la versión original es de $costeoptimo = 6 * 3 + 3 + 0 = 21$.

Una posible solución con este coste para este problema sería la siguiente:

```
 0:  MAKE_SANDWICH_NO_GLUTEN SANDW1 BREAD2 CONTENT3
1:  MAKE_SANDWICH_NO_GLUTEN SANDW2 BREAD5 CONTENT6
2:  MAKE_SANDWICH SANDW3 BREAD1 CONTENT1
3:  MAKE_SANDWICH SANDW4 BREAD3 CONTENT2
4:  MAKE_SANDWICH SANDW5 BREAD4 CONTENT4
5:  MAKE_SANDWICH SANDW6 BREAD6 CONTENT5
6:  PUT_ON_TRAY SANDW1 TRAY1
7:  PUT_ON_TRAY SANDW2 TRAY1
8:  PUT_ON_TRAY SANDW3 TRAY1
```

```
9:  PUT_ON_TRAY SANDW4 TRAY1
10:PUT_ON_TRAY SANDW5 TRAY1
11:PUT_ON_TRAY SANDW6 TRAY1
12:MOVE_TRAY TRAY1 KITCHEN TABLE1
13:SERVE_SANDWICH_NO_GLUTEN SANDW1 CHILD5 TRAY1 TABLE1
14:SERVE_SANDWICH SANDW3 CHILD2 TRAY1 TABLE1
15:SERVE_SANDWICH SANDW4 CHILD6 TRAY1 TABLE1
16:MOVE_TRAY TRAY1 TABLE1 TABLE2
17:SERVE_SANDWICH_NO_GLUTEN SANDW2 CHILD1 TRAY1 TABLE2
18:SERVE_SANDWICH SANDW5 CHILD4 TRAY1 TABLE2
19:MOVE_TRAY TRAY1 TABLE2 TABLE3
20:SERVE_SANDWICH SANDW6 CHILD3 TRAY1 TABLE3
```

Por último, si se observa detenidamente esta solución, se podrá ver que no hay ningún conjunto de acciones que se pueda realizar en un menor número de pasos. Todos los sándwiches tienen que hacerse, por lo que hasta el paso 5 no hay ninguna reducción de costes posibles. Todos los sándwiches tienen que ser cargados en alguna bandeja, porque sólo se pueden servir sándwiches cargados, por lo que los pasos 6-11 no pueden ser simplificados. Es necesario una acción por cada sándwich que se sirva y no se puede servir un sándwich si no se encuentra la bandeja que contiene ese sándwich en la misma localización que el niño al que se va a servir, por lo que tampoco existe ninguna manera de reducir el número de acciones en el tramo 13-20. Por tanto, con este ejemplo empírico queda patente que la fórmula utilizada es correcta.

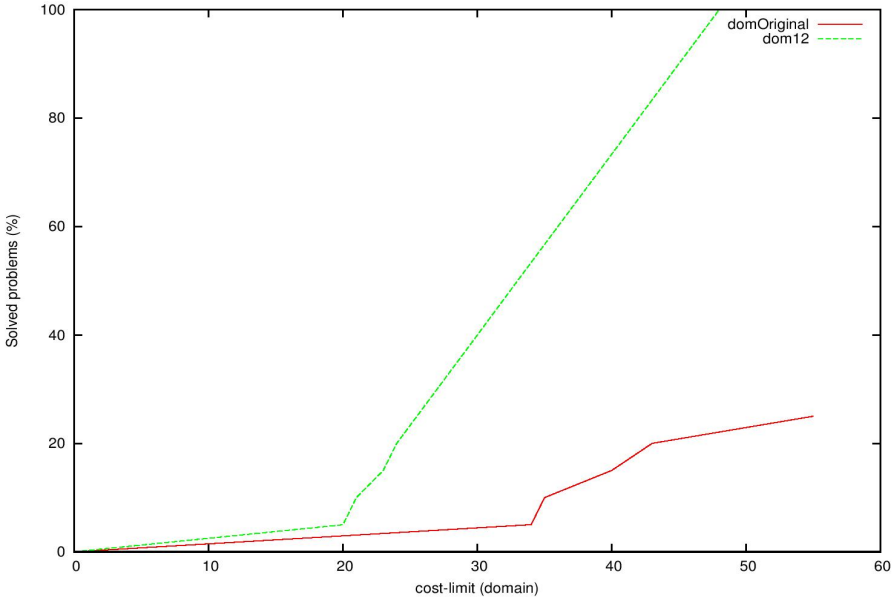

Figura 21: MFF-original-12-coste

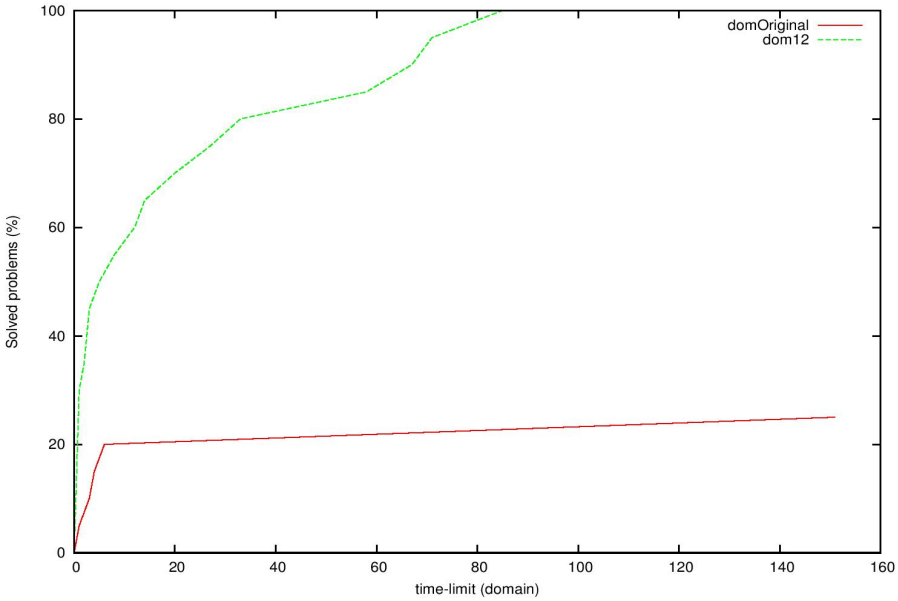

Figura 22: MFF-original-12-tiempo

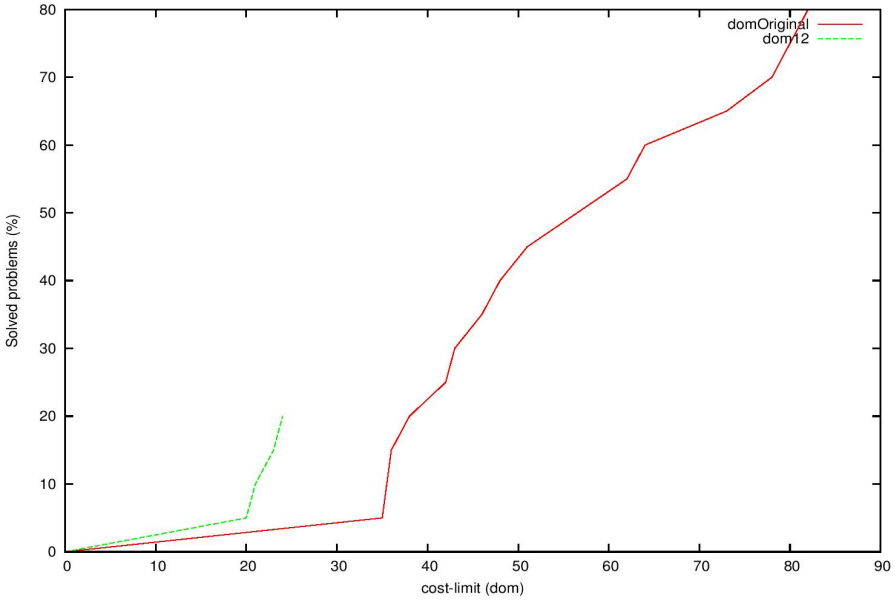

Figura 23: LPG-original-12-coste

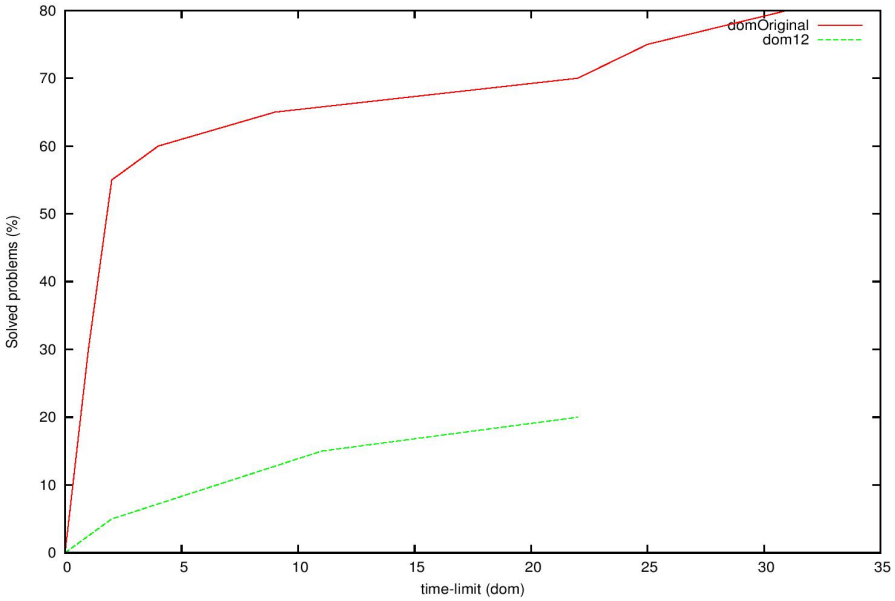

Figura 24: LPG-original-12-tiempo

Tabla 2: Coste óptimo y coste de MFF

Problema	**Coste óptimo**	**Coste MFF**
01-2	21	**21**
01	20	**20**
02-2	24	**24**
02	22	23
03-2	27	**27**
03	27	**27**
04-2	30	**30**
04	30	**30**
05-2	33	**33**
05	33	**33**
06-2	36	**36**
06	36	**36**
07-2	39	**39**
07	39	**39**
08-2	39	42
08	39	42
09-2	45	**45**
09	45	**45**
10-2	48	**48**
10	48	**48**

Como se puede ver en las figuras 21, 22, 23 y 24, vuelve a haber una gran diferencia entre las gráficas obtenidas con *MFF* y con *LPG-td*. En el caso de las figuras 21 y 22, se puede ver que esta modificación mejora muy significativamente tanto los tiempos de búsqueda como la calidad de las soluciones encontradas, hasta el punto de que como se puede observar en la tabla superior, en 17 de los 20 problemas originales el coste de la solución obtenida es el mismo que el menor que se podría obtener en cualquier versión que se hiciese de este problema, lo que convierte a esta versión en la mejor de todas las creadas.

Por contra, se puede observar en las figuras 23 y 24 que los resultados obtenidos utilizando el planificador *LPG-td* son bastante peores que los obtenidos por la versión original en coste, tiempo de resolución y porcentaje de problemas resueltos. Una vez más, sería interesante poder analizar con mayor profundidad los motivos por los que se han obtenido estos resultados con el planificador *LPG-td*, pero al no haber sido capaz de entender bien el funcionamiento de este planificador, no se puede explicar mejor los resultados que se han obtenido tras ejecutar este planificador

7.3.6 Comparaciones globales

En esta sección, se va a incluir una comparativa de los resultados referentes a los tiempos y los costes de resolución de cada uno de los problemas para cada planificador por cada una de las versiones del dominio. De esta manera, se permitirá tener una visión más global de los resultados obtenidos. Las fórmulas que se utilizarán

para esta comparativa son las mismas que la que se utilizan en las competiciones de la *IPC* y son las siguientes:

$$puntuación_a(tiempo) = \begin{cases} 0 & \text{si } a.E = -1 \\ \frac{1}{1+log_{10}(\frac{a.D}{a.F})} & \text{si } a.E \neq -1 \end{cases}$$

$$puntuación_a(calidad) = \begin{cases} 0 & \text{si } a.E = -1 \\ \frac{1}{1+log_{10}(\frac{a.E}{a.G})} & \text{si } a.E \neq -1 \end{cases}$$

Donde a se define como la siguiente tupla:

$a = (A, B, C, D, E, F, G)$

$A =$ problema actual.

$B =$ planificador utilizado.

$C =$ tiempo límite de la planificación.

$D =$ tiempo de búsqueda de la solución.

$E =$ coste de la solución (-1 si no ha sido resuelto).

$F =$ mejor tiempo de resolución del problema$A$entre todas las versiones del problema.

$G =$ menor coste de resolución del problema$A$entre todas las versiones del problema.

Aplicando esta fórmula para cada uno de los planificadores, obtenemos las tablas 3, 4, 5, 6 y 7 que observamos a continuación.

Tabla 3: MFF tiempos *IPC*

problema / versión	**original**	**01**	**02**	**03**	**05**	**08**	**09**	**11**	**12**	**numProps**	**numDigits**
p01-2	0,677	1	0,624	1	1	0	0,624	0,624	1	1	1
p01	0,677	1	0,624	0	1	0	0,624	0,624	1	1	0
p02-2	0,624	1	0,624	0,235	1	0	0,677	0,624	1	1	0
p02	0,315	1	0,308	0	1	0,235	0,318	0,318	1	1	0,235
p03-2	0	1	0	0	1	0	0	0	1	0,769	0
p03	0,235	1	0,235	0	0,769	0,235	0,235	0,235	1	0,589	0,235
p04-2	0	1	0	0	0,769	0	0	0	1	0,624	1
p04	0	1	0	0	0,542	0	0	0	0,677	0,500	0
p05-2	0	1	0	0,235	0,542	0	0	0	0,769	0,500	0
p05	0	1	0	0	0,466	0	0	0	0,589	0,439	0
p06-2	0	1	0	0	0,454	0	0	0	0,677	0,439	0
p06	0	1	0	0	0,420	0	0	0	0,481	0,373	0
p07-2	0	1	0	0	0,414	0	0	0	0,525	0,374	0
p07	0	1	0	0	0,473	0	0	0	0,548	0,402	0
p08-2	0	1	0	0	0,435	0	0	0	0,542	0,375	0
p08	0	1	0	0	0,423	0	0	0	0,547	0,391	0,274
p09-2	0	0,624	0	0	0,335	0	0	0	0,397	0,316	0
p09	0	1	0	0	0,409	0	0	0	0,548	0,409	0
p10-2	0	1	0	0	0	0	0	0	0,553	0,402	0
p10	0	1	0	0	0	0	0	0	0,614	0,433	0
Media	**0,126**	**0,981**	**0,121**	**0,074**	**0,572**	**0,024**	**0,124**	**0,121**	**0,723**	**0,567**	**0,137**

Tabla 4: MFF calidad *IPC*

problema/versión	original	01	02	03	05	08	09	11	12	numProps	numDigits
p01-2	0,751	0,885	0,751	1,000	0,873	0,000	0,751	0,751	0,979	0,960	0,873
p01	0,813	0,885	0,813	0,000	0,873	0,000	0,813	0,861	1,000	0,960	0,000
p02-2	0,735	0,889	0,735	1,000	0,879	0,000	0,735	0,735	1,000	0,966	0,000
p02	0,846	0,875	0,846	0,000	0,864	0,846	0,846	0,846	1,000	0,949	0,885
p03-2	0,000	0,880	0,000	0,000	0,871	0,000	0,000	0,000	1,000	0,956	0,000
p03	0,854	0,880	0,854	0,000	0,871	0,854	0,854	0,854	1,000	0,956	0,871
p04-2	0,000	0,881	0,000	0,000	0,873	0,000	0,000	0,000	1,000	0,960	0,873
p04	0,000	0,881	0,000	0,000	0,873	0,000	0,000	0,000	1,000	0,960	0,000
p05-2	0,000	0,874	0,000	0,964	0,867	0,000	0,000	0,000	1,000	0,953	0,000
p05	0,000	0,874	0,000	0,000	0,867	0,000	0,000	0,000	1,000	0,953	0,000
p06-2	0,000	0,869	0,000	0,000	0,862	0,000	0,000	0,000	1,000	0,947	0,000
p06	0,000	0,869	0,000	0,000	0,862	0,000	0,000	0,000	1,000	0,947	0,000
p07-2	0,000	0,864	0,000	0,000	0,859	0,000	0,000	0,000	1,000	0,941	0,000
p07	0,000	0,864	0,000	0,000	0,859	0,000	0,000	0,000	1,000	0,941	0,000
p08-2	0,000	0,861	0,000	0,000	0,855	0,000	0,000	0,000	1,000	0,937	0,000
p08	0,000	0,861	0,000	0,000	0,855	0,000	0,000	0,000	1,000	0,937	0,855
p09-2	0,000	0,857	0,000	0,000	0,853	0,000	0,000	0,000	1,000	0,934	0,000
p09	0,000	0,857	0,000	0,000	0,853	0,000	0,000	0,000	1,000	0,934	0,000
p10-2	0,000	0,855	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,931	0,000
p10	0,000	0,855	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,931	0,000
Media	**0,200**	**0,871**	**0,200**	**0,148**	**0,778**	**0,085**	**0,200**	**0,202**	**0,999**	**0,948**	**0,218**

Tabla 5: LPG-td tiempos *IPC*

problema/versión	**original**	**01**	**02**	**03**	**05**	**08**	**09**	**11**	**12**	**numProps**	**numDigits**
p01-2	1	1	1	1	0,250	1	1	1	0,490	0	0,331
p01	1	1	1	1	0,303	1	1	1	0,769	0,466	0,335
p02-2	1	1	1	1	0	1	1	1	0,427	0	0,319
p02	1	1	1	1	0	1	1	1	0,490	0	0,334
p03-2	1	1	1	1	0,244	1	1	1	0	0	0,320
p03	1	0,769	1	1	0,244	1	1	1	0	0	0,323
p04-2	0,769	0,769	0,769	0,677	0	1	1	1	0	0	0,317
p04	0,769	1	1	0,677	0	1	1	1	0	0	0,313
p05-2	0,769	0,624	0,542	0,542	0	1	1	1	0	0	0,312
p05	0,769	0,769	1	0,542	0	1	1	1	0	0	0,287
p06-2	0	0,500	0,460	0,435	0	1	1	1	0	0	0,326
p06	0,769	0,677	0,525	0,435	0	1	1	1	0	0	0,293
p07-2	0	0	0,624	0,366	0	1	1	1	0	0	0,289
p07	0,624	0,490	0,466	0,366	0	1	1	1	0	0	0,323
p08-2	0,401	0	0,401	1	0	1	1	1	0	0	0,297
p08	0,417	0,624	0,589	1	0	1	0,624	0,677	0	0	0,314
p09-2	0,427	0	0	1	0	1	0,624	0,677	0	0	0,250
p09	0,512	0,454	0,490	1	0	1	1	0,542	0	0	0,252
p10-2	0	0	0,349	1	0	1	0,677	1	0	0	0,282
p10	0	0	0,589	1	0	1	0,769	0,443	0	0	0,266
Media	**0,611**	**0,584**	**0,690**	**0,802**	**0,052**	**1**	**0,935**	**0,917**	**0,109**	**0,023**	**0,304**

Tabla 6: LPG-td calidad *IPC*

problemaón	**original**	**01**	**02**	**03**	**05**	**08**	**09**	textbf11	**12**	**numProps**	**numDigits**
p01-2	0,810	0,915	0,855	0,945	0,902	0,781	0,930	0,962	1	0	0,845
p01	0,797	0,898	0,943	0,927	0,912	0,813	0,912	0,979	1	1	0,797
p02-2	0,769	0,912	0,859	0,937	0	0,818	0,900	0,834	1	0	0,826
p02	0,846	0,885	0,864	0,921	0	0,813	0,965	0,935	1	0	0,799
p03-2	0,931	0,974	0,931	1	0,987	0,838	0,974	0,974	0	0	0,903
p03	0,886	0,963	0,903	1	0,974	0,894	1	0,963	0	0	0,832
p04-2	0,884	0,925	0,907	0,976	0	0,793	0,976	1	0	0	0,798
p04	0,916	0,944	0,891	0,976	0	0,850	1	0,934	0	0	0,891
p05-2	0,905	0,979	0,943	1	0	0,891	0,935	0,989	0	0	0,879
p05	0,840	0,970	0,960	1	0	0,856	1	0,970	0	0	0,817
p06-2	0	0,947	0,964	1	0	0,865	0,947	1	0	0	0,860
p06	0,870	0,947	0,981	1	0	0,870	0,955	0,990	0	0	0,870
p07-2	0	0	0,867	0,973	0	0,907	1	0,956	0	0	0,787
p07	0,846	0,959	0,959	1	0	0,884	0,966	0,991	0	0	0,864
p08-2	0,835	0	0,876	1	0	0,793	0,955	0,955	0	0	0,839
p08	0,917	0,992	0,906	1	0	0,876	0,976	0,976	0	0	0,824
p09-2	0,866	0	0	1	0	0,829	0,978	1	0	0	0,887
p09	0,862	0,950	0,875	0,984	0	0,781	1	0,909	0	0	0,858
p10-2	0	0	0,840	1	0	0,876	1	0,979	0	0	0,861
p10	0	0	0,932	1	0	0,760	0,973	0,854	0	0	0,700
Media	**0,689**	**0,708**	**0,863**	**0,982**	**0,189**	**0,839**	**0,967**	**0,958**	**0,200**	**0,050**	**0,837**

Tabla 7: Puestos valores *IPC*

Versión	Puesto tiempo MFF	Puesto calidad MFF	Puesto total MFF[1]	Puesto tiempo LPG-td	Puesto calidad LPG-td	Puesto total LPG-td[1]
Original	6º	7º	6º	6º	8º	6º
01	1º	3º	1º	7º	7º	7º
02	8º	7º	9º	5º	4º	5º
03	10º	10º	10º	4º	1º	4º
05	3º	4º	4º	10º	10º	10º
08	11º	11º	11º	1º	5º	3º
09	7º	7º	7º	2º	2º	1º
11	8º	6º	7º	3º	3º	2º
12	2º	1º	2º	9º	9º	9º
numProps	4º	2º	3º	11º	11º	11º
numDigits	5º	5º	5º	8º	6º	8º

[1]Obtenido de la comparación de la suma de los valores de tiempo y calidad

Como se puede ver, existe gran diferencia entre los resultados obtenidos cuando el planificador varía. Para ser más precisos, esta variación se puede medir con el **coeficiente de correlación de Pearson**, que permite cuantificar la diferencia entre dos conjuntos numéricos independiente de la escala de medida de estos conjuntos. La fórmula para hallar la correlación de Pearson es la siguiente:

$$r = \frac{\sum_i^n (X_i - \overline{X}) * ((Y_i - \overline{Y}))}{\sqrt{\sum_i^n (X_i - \overline{X})^2 * \sum_i^n (Y_i - \overline{Y})^2}}$$

Donde X_i e Y_i son los respectivos valores de cada una de las variables que se quiere comparar en cada posición de cada uno de estos conjuntos, $\overline{X}$ e $\overline{Y}$ el valor medio de cada uno de estos conjuntos. El valor r que se obtiene está comprendido en $[-1, 0]$. Los valores que se acercan a 1 implican una correlación positiva. Los valores que se acercan a -1 una correlación negativa. Los valores cercanos a 0 implican una falta de correlación entre ambos conjuntos.

El coeficiente de correlación de Pearson obtenido entre los tiempos de búsqueda de las versiones ejecutadas con *MFF* y y *LPG-td* es de **-0,6256**. Esto implica una moderada correlación negativa entre los tiempos de búsqueda de ambos planificadores, es decir, en términos generales, las versiones que obtienen buenos resultados temporales obtienen tiempos peores con el otro planificador sin llegar a tener una correlación en la que el mejor desempeño con una de las versiones signifique el peor desempeño con el otro planificador. Estos son resultados de una muestra de únicamente 11 representaciones, por lo que no es posible sacar conclusiones más genéricas respecto a la correlación en el desempeño temporal entre ambos planificadores. Sin embargo, sí es posible determinar que a priori va a resultar complicado encontrar para este dominio una versión que pueda ser resuelta en el menor tiempo posible por ambos planificadores. De hecho, la segunda versión más rápida obtenida con el planificador *MFF* es la que obtiene el segundo peor resultado temporal con *LPG-td*. A su vez, la versión que resuelve los problemas en un menor tiempo con *LPG-td* obtiene el peor resultado temporal al cambiar el planificador a *MFF*.

Respecto a los resultados globales, se puede ver que el acotamiento del árbol de búsqueda que supone la modificación del dominio 01 ha supuesto grandes ventajas en términos de tiempo de búsqueda con el planificador *MFF*, sin que esto suponga grandes pérdidas en términos de calidad de las soluciones obtenidas. Esto sin embargo se debe a las particularidades de los problemas de entrada: si cada uno de los niños a los que fuera necesario servir se encontrase en una posición distinta y el número de ellos fuera elevado, la calidad de la solución se vería ampliamente afectada.

Por otro lado, la versión 12 ha sido creada explícitamente pensando tanto en el tiempo como en la calidad de la solución para el planificador *MFF*. Es por ello que obtiene el mejor puesto en cuanto a la calidad (ya que como se ha comentado previamente, utiliza predicados que impiden que se realicen acciones que empeorarían la calidad de la solución) a la vez que se obtiene un segundo puesto en el tiempo de búsqueda, puesto que el hecho de que obtenga una solución de menor coste sin que el dominio le permita explorar otras ramas también implica un menor tiempo de búsqueda. De hecho, como se puede ver en los resultados tanto de *MFF* como de *LPG-td*, existe una correlación muy clara entre la calidad y el tiempo de búsqueda del planificador.

Por último, se puede ver que en ambos casos se ha conseguido obtener un conjunto de dominios que mayoritariamente ha tenido un desempeño mejor que la versión original, por lo que se puede determinar que existe un gran margen de mejora en este dominio con el fin de reducir los tiempos de búsqueda y mejorar la calidad de la solución con diversos planificadores.

8 Conclusiones y trabajo futuro

Tras la realización de este trabajo, en esta sección se van a incluir las conclusiones que se han extraído y las futuras líneas de trabajo en las que se puede continuar el desarrollo de este proyecto.

8.1 Conclusiones

En este trabajo se ha podido verificar que las hipótesis iniciales eran correctas: por un lado es posible generar varias representaciones para un mismo problema, y por otro, la manera en que se representan los problemas influyen significativamente en los resultados obtenidos en la planificación automática.

A su vez, como se ha visto, estos resultados son muy dependientes del tipo de planificador. Las modificaciones manuales se han adaptado mejor al planificador *MFF*, pues es mucho más intuitivo y fácil de comprender que el otro planificador utilizado, *LPG-td*.

Respecto a los resultados obtenidos con el software de terceros desarrolladores, los resultados obtenidos con aquellos que cambiaban la representación de los dominios a numérico con dígitos o con palabras han sido bastante aceptables, siendo la versión numérica con dígitos prácticamente idéntica a la versión realizada a mano. Por otro lado, el software *PTT* que permite generar macrooperadores no ha sido capaz de encontrar ningún macroooperador a pesar de demostrar en las versiones realizadas a mano que existían múltiples opciones que además mejoran el proceso de planificación.

Como las pruebas se han realizado casi exclusivamente en un único dominio, la generalización de estos resultados debe realizarse con cautela. Estos resultados permiten extraer la conclusión de que la variación de la representación de los objetos influye en el proceso de planificación para cada tipo de planificador. Sin embargo, no es posible extraer conclusiones concluyentes referentes a la utilidad del software de terceros desarrolladores utilizados en este trabajo ya que para eso hubiese sido necesario realizar estas pruebas con un abanico mucho más amplio de dominios. Sin embargo, a la vista de los resultados obtenidos, sí es posible concluir que en algunos problemas como en *Childsnack* existe un amplio margen de mejoras en relación a los dominios generados por el software utilizado, lo que lleva a sospechar que esto puede suceder en otros dominios.

Por último, los objetivos que se han marcado para este trabajo han sido alcanzados en tanto en cuanto se ha realizado el estudio tal y como se ha mencionado y se considera que este trabajo puede ayudar a la consecución de la obtención de mejores representaciones de problemas de planificación automática.

De manera más personal, este trabajo me ha permitido realizar una actividad enormemente estimulante como es poder estar en contacto con muchos investigadores de Europa que están actualmente trabajando en planificación automática y que me han resuelto bastantes dudas que tenía referentes al propio software que habían desarrollado. Sin ir más lejos, El software de PTT contenía una errata en la documentación según la cual uno de los parámetros de entrada era "-p" en lugar de "-q". Gracias al contacto con este desarrollador, él pudo actualizar la documentación para evitar que otras personas cayesen en el mismo error que yo y yo pude utilizar su software.

Creo que este trabajo a su vez me ha aportado la posibilidad de investigar lo que se está realizando en un área de estudio con el fin real de entender lo que se ha conseguido, a diferencia de lo que puede ocurrir en otros proyectos enfocados al desarrollo cuyo único fin puede ser rellenar una sección de la memoria llamada "estado del arte". Creo que esto ha sido una oportunidad para poder aportar un granito de arena a este área de investigación y por ello personalmente me alegro de haber podido desarrollar este trabajo en lugar de otros que tuviesen como único fin de tener aprobados los 12 créditos correspondientes a esta atípica asignatura.

8.2 Trabajo futuro

En este trabajo, debido a la complejidad del planificador LPG-td, las conclusiones extraídas tienen bastante margen de mejora. Existe una gran diferencia entre las explicaciones de los resultados obtenidos con el planificador MFF respecto a las conclusiones de los resultados obtenidos con LPG-td. Por lo tanto, lo primero que se podría hacer para continuar con este proyecto sería entender mejor el funcionamiento de LPG-td para así poder dar mejores explicaciones de los resultados obtenidos.

Tras esto, se pueden añadir más planificadores con los que realizar pruebas a los dominios utilizados. Se han utilizado únicamente dos planificadores porque se ha intentado centrar los esfuerzos en la generación de distintas versiones de los problemas más que en los resultados obtenidos para cada tipo de planificador, por lo que este aspecto puede ser mejorado.

Por último, la parte más costosa pero a la vez la que más útil podría ser sería intentar extrapolar los resultados obtenidos por cada tipo de planificador que se tome en consideración para cada tipo de modificación del dominio que sea extrapolable a otros dominios completamente distinto. Esto permitiría generar unas pautas para la modificación de los dominios que permitiesen la mejora de los dominios para cada planificador. Inicialmente esto podría ser simplemente unas pautas para los creadores de problemas en *PDDL* pero más adelante se podría transformar en un software independiente de dominio que identificase las pautas de las distintas modificaciones posibles del dominio y las ejecutase con el fin de acercarse a la versión más óptima del mismo para un tipo de planificador concreto midiendo la optimalidad en función del coste y del tiempo de búsqueda necesario del planificador.

Se es consciente de que esta última línea de investigación futura es de una enorme complejidad pero tal y como se ha demostrado con las modificaciones realizadas a mano en esta memoria, esta complejidad se podría ver recompensada con la generación de dominios que mejorasen sustancialmente tanto en el coste como principalmente en el tiempo de planificación.

9 Anexos

9.1 Anexo I: Dominios utilizados y problemas de entrada

En esta sección se va a incluir el código de cada una de las versiones de *Childsnack* descritas en la sección de solución propuesta así como el problema de entrada 01-2 correspondiente a la versión original versionado para cada uno de los nuevos dominios. Como para muchas de las versiones las modificaciones que se realizan son bastante pequeñas respecto a alguna de las otras versiones, para evitar la reiteración en lugar de incluir el código completo de cada versión, se van a señalar las modificaciones realizadas entre las distintas versiones para así incluir sólo aquella parte en la que varíe la sección que se está exponiendo.

9.1.1 Versión original

9.1.1.1 Dominio

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; The child-snack domain 2013
;;
;; This domain is for planning how to make and serve sandwiches for a group of
;; children in which some are allergic to gluten. There are two actions for
;; making sandwiches from their ingredients. The first one makes a sandwich and
;; the second one makes a sandwich taking into account that all ingredients are
;; gluten-free. There are also actions to put a sandwich on a tray, to move a tray
;; from one place to another and to serve sandwiches.
;;
;; Problems in this domain define the ingredients to make sandwiches at the initial
;; state. Goals consist of having all kids served with a sandwich to which they
;; are not allergic.
;;
;; Author: Raquel Fuentetaja and Tom s de la Rosa
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define (domain child-snack)
    (:requirements :typing :equality)
    (:types child bread-portion content-portion sandwich tray place)
    (:constants kitchen - place)

    (:predicates
        (at_kitchen_bread ?b - bread-portion)
        (at_kitchen_content ?c - content-portion)
        (at_kitchen_sandwich ?s - sandwich)
        (no_gluten_bread ?b - bread-portion)
        (no_gluten_content ?c - content-portion)
        (ontray ?s - sandwich ?t - tray)
        (no_gluten_sandwich ?s - sandwich)
        (allergic_gluten ?c - child)
        (not_allergic_gluten ?c - child)
        (served ?c - child)
        (waiting ?c - child ?p - place)
        (at ?t - tray ?p - place)
        (notexist ?s - sandwich))

```

```
41      (:action make_sandwich_no_gluten
42          :parameters (?s - sandwich ?b - bread-portion ?c - content-portion)
43
44          :precondition (and
45              (at_kitchen_bread ?b)
46              (at_kitchen_content ?c)
47              (no_gluten_bread ?b)
48              (no_gluten_content ?c)
49              (notexist ?s))
50
51          :effect (and
52              (not (at_kitchen_bread ?b))
53              (not (at_kitchen_content ?c))
54              (at_kitchen_sandwich ?s)
55              (no_gluten_sandwich ?s)
56              (not (notexist ?s)))
57      )
58
59
60      (:action make_sandwich
61          :parameters (?s - sandwich ?b - bread-portion ?c - content-portion)
62
63          :precondition (and
64              (at_kitchen_bread ?b)
65              (at_kitchen_content ?c)
66              (notexist ?s))
67
68          :effect (and
69              (not (at_kitchen_bread ?b))
70              (not (at_kitchen_content ?c))
71              (at_kitchen_sandwich ?s)
72              (not (notexist ?s)))
73      )
74
75
76      (:action put_on_tray
77          :parameters (?s - sandwich ?t - tray)
78
79          :precondition (and
80              (at_kitchen_sandwich ?s)
81              (at ?t kitchen))
82
83          :effect (and
84              (not (at_kitchen_sandwich ?s))
85              (ontray ?s ?t))
86      )
87
88
89      (:action serve_sandwich_no_gluten
90          :parameters (?s - sandwich ?c - child ?t - tray ?p - place)
91
92          :precondition (and
93              (allergic_gluten ?c)
94              (ontray ?s ?t)
95              (waiting ?c ?p)
96              (no_gluten_sandwich ?s)
97              (at ?t ?p))
98
99          :effect (and
100             (not (ontray ?s ?t))
101             (served ?c))
102     )
103
104     (:action serve_sandwich
105         :parameters (?s - sandwich ?c - child ?t - tray ?p - place)
```

```
106
107         :precondition (and
108             (not_allergic_gluten ?c)
109             (waiting ?c ?p)
110             (ontray ?s ?t)
111             (at ?t ?p))
112
113         :effect (and
114             (not (ontray ?s ?t))
115             (served ?c))
116     )
117
118     (:action move_tray
119         :parameters (?t - tray ?p1 ?p2 - place)
120
121         :precondition (and (at ?t ?p1))
122
123         :effect (and
124             (not (at ?t ?p1))
125             (at ?t ?p2))
126     )
127
128 )
```

9.1.1.2 Problema

```
1  ; child-snack task with 6 children and 0.4 gluten factor
2  ; constant factor of 1.3
3  ; random seed: 234324
4
5  (define (problem prob-snack)
6    (:domain child-snack)
7    (:objects
8      child1 child2 child3 child4 child5 child6 - child
9      bread1 bread2 bread3 bread4 bread5 bread6 - bread-portion
10     content1 content2 content3 content4 content5 content6 - content-portion
11     tray1 tray2 - tray
12     table1 table2 table3 - place
13     sandw1 sandw2 sandw3 sandw4 sandw5 sandw6 sandw7 sandw8 - sandwich
14   )
15   (:init
16      (at tray1 kitchen)
17      (at tray2 kitchen)
18      (at_kitchen_bread bread1)
19      (at_kitchen_bread bread2)
20      (at_kitchen_bread bread3)
21      (at_kitchen_bread bread4)
22      (at_kitchen_bread bread5)
23      (at_kitchen_bread bread6)
24      (at_kitchen_content content1)
25      (at_kitchen_content content2)
26      (at_kitchen_content content3)
27      (at_kitchen_content content4)
28      (at_kitchen_content content5)
29      (at_kitchen_content content6)
30      (no_gluten_bread bread2)
31      (no_gluten_bread bread5)
32      (no_gluten_content content3)
33      (no_gluten_content content6)
34      (allergic_gluten child1)
35      (allergic_gluten child5)
36      (not_allergic_gluten child2)
37      (not_allergic_gluten child3)
38      (not_allergic_gluten child4)
39      (not_allergic_gluten child6)
```

```
40        (waiting child1 table2)
41        (waiting child2 table1)
42        (waiting child3 table3)
43        (waiting child4 table2)
44        (waiting child5 table1)
45        (waiting child6 table1)
46        (notexist sandw1)
47        (notexist sandw2)
48        (notexist sandw3)
49        (notexist sandw4)
50        (notexist sandw5)
51        (notexist sandw6)
52        (notexist sandw7)
53        (notexist sandw8)
54     )
55     (:goal
56       (and
57        (served child1)
58        (served child2)
59        (served child3)
60        (served child4)
61        (served child5)
62        (served child6)
63       )
64     )
65  )
```

9.1.2 Versión 01

9.1.2.1 Dominio

Los cambios de representación son los siguientes:

- Juntar operadores `move_tray` con los operadores `serve_sandwich` y `serve_sandwich_no_gluten` , produciendo por tanto los macrooperadores `move_tray_serve_sandwich_no_gluten` y `move_tray_serve_sandwich` .
- Incluir predicado numérico con dígitos `total_cost` que se incrementa en una unidad en cada operador inicial, menos en los macrooperadores citados en el punto anterior que se incrementan en dos unidades.

```
(:action move_tray_serve_sandwich_no_gluten
    :parameters (?t - tray ?p1 ?p2 - place ?s - sandwich ?c - child)

    :precondition (and
        (at ?t ?p1)
        (waiting ?c ?p2)
        (ontray ?s ?t)
        (allergic_gluten ?c)
        (no_gluten_sandwich ?s))

    :effect (and
        (not (at ?t ?p1))
        (at ?t ?p2)
        (not (ontray ?s ?t))
        (served ?c)
        (increase (total-cost) 2))
)
```

```
(:action move_tray_serve_sandwich
    :parameters (?t - tray ?p1 ?p2 - place ?s - sandwich ?c - child)
    :precondition (and
        (at ?t ?p1)
        (waiting ?c ?p2)
        (ontray ?s ?t))

    :effect (and
        (not (at ?t ?p1))
        (at ?t ?p2)
        (not (ontray ?s ?t))
        (served ?c)
        (increase (total-cost) 1))
)
```

9.1.2.2 Problema

Lo único que ha cambiado en la definición del dominio respecto al dominio original es que se ha inicializado la variable `total_cost` a cero y se ha incluido como métrica minimizar el valor de `total_cost` .

```
(= (total-cost) 0)
```

```
(:metric minimize (total-cost))
```

9.1.3 Versión 02

9.1.3.1 Dominio

La representación del dominio varía únicamente en la definición del operador `make_sandwich` .

```
(:action make_sandwich

    :parameters (?s - sandwich ?b - bread-portion ?c - content-portion)

    :precondition (and
        (at_kitchen_bread ?b)
        (at_kitchen_content ?c)
        (notexist ?s)
        (not (and (no_gluten_bread ?b) (no_gluten_content ?c))))

    :effect (and
        (not (at_kitchen_bread ?b))
        (not (at_kitchen_content ?c))
        (at_kitchen_sandwich ?s)
        (not (notexist ?s)))
)
```

9.1.3.2 Problema

La definición de los problemas de esta versión es idéntica a la definición de la versión original.

9.1.4 Versión 03

9.1.4.1 Dominio

El dominio de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el dominio íntegro.

```
1  (define (domain child-snack)
2      (:requirements :typing :equality)
3      (:types child bread-portion content-portion sandwich tray place)
4      (:constants kitchen - place)
5
6      (:predicates
7          (at_kitchen_bread ?b - bread-portion)
8          (at_kitchen_content ?c - content-portion)
9          (at_kitchen_sandwich ?s - sandwich)
10         (no_gluten_bread ?b - bread-portion)
11         (no_gluten_content ?c - content-portion)
12         (ontray ?s - sandwich ?t - tray)
13         (no_gluten_sandwich ?s - sandwich)
14         (allergic_gluten ?c - child)
15         (not_allergic_gluten ?c - child)
16         (served ?c - child)
17         (waiting ?c - child ?p - place)
18         (at ?t - tray ?p - place)
19         (notexist ?s - sandwich))
20
21     (:functions (total-cost) - number)
22
23     (:action make_sandwich_move_tray_serve_sandwich_move_tray_kitchen
24         :parameters (?t - tray ?p2 - place ?s - sandwich ?c - child ?b - bread-portion
25         ?ct - content-portion)
26
27         :precondition (and
28             (at ?t kitchen)
29             (at_kitchen_bread ?b)
30             (at_kitchen_content ?ct)
31             (notexist ?s)
32             (waiting ?c ?p2)
33             (not_allergic_gluten ?c))
34
35         :effect (and
36             (not (at_kitchen_bread ?b))
37             (not (at_kitchen_content ?ct))
38             (not (at ?t kitchen))
39             (at ?t kitchen)
40             (not (ontray ?s ?t))
41             (not (notexist ?s))
42             (served ?c))
43             (increase (total-cost) 4)
44     )
45
46     (:action make_sandwich_move_tray_serve_sandwich_no_gluten_move_tray_kitchen
47         :parameters (?t - tray ?p2 - place ?s - sandwich ?c - child ?b - bread-portion
48         ?ct - content-portion)
49
50         :precondition (and
```

```
51              (at ?t kitchen)
52              (at_kitchen_bread ?b)
53              (at_kitchen_content ?ct)
54              (notexist ?s)
55              (waiting ?c ?p2)
56              (allergic_gluten ?c)
57              (no_gluten_bread ?b)
58              (no_gluten_content ?ct))
59
60          :effect (and
61              (not (at_kitchen_bread ?b))
62              (not (at_kitchen_content ?ct))
63              (not (at ?t kitchen))
64              (at ?t kitchen)
65              (not (ontray ?s ?t))
66              (not (notexist ?s))
67              (served ?c))
68              (increase (total-cost) 4)
69      )
70
71      (:action move_tray
72          :parameters (?t - tray ?p1 ?p2 - place)
73
74          :precondition (and (at ?t ?p1))
75
76          :effect (and
77              (not (at ?t ?p1))
78              (at ?t ?p2))
79              (increase (total-cost) 1)
80      )
81  )
```

9.1.4.2 Problema

La definición de los problemas de esta versión es idéntica a la definición de la versión 01.

9.1.5 Versión 05

9.1.5.1 Dominio

El dominio de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el dominio íntegro de esta versión.

```
1   (define (domain child-snack)
2       (:requirements :typing :equality)
3       (:types num tray place )
4       (:constants kitchen - place)
5
6       (:predicates
7           (at_kitchen_bread ?gluten - num ?no-gluten - num)
8           (at_kitchen_content ?gluten - num ?no-gluten - num)
9           (at_kitchen_sandwich ?gluten - num ?no-gluten - num)
10          (at ?t - tray ?p - place)
11          (next ?minor - num ?major - num)
12          (ontray ?gluten - num ?no-gluten - num ?t - tray)
13          (served_childs ?served - num)
14          (allergic_gluten ?c - child)
15          (not_allergic_gluten ?c - child)
16          (waiting ?c - child ?p - place))
17
```

```
18
19      (:action make_sandwich_no_gluten
20          :parameters (?bread-gluten ?bread-no-gluten ?content-gluten ?content-no-gluten
21          ?sandwich-gluten ?sandwich-no-gluten ?bread-no-gluten-minor ?content-no-gluten-minor
22          ?sandwich-no-gluten-major - num)
23
24          :precondition (and
25              (at_kitchen_bread ?bread-gluten ?bread-no-gluten)
26              (at_kitchen_content ?content-gluten ?content-no-gluten)
27              (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
28              (next ?bread-no-gluten-minor ?bread-no-gluten)
29              (next ?content-no-gluten-minor ?content-no-gluten)
30              (next ?sandwich-no-gluten ?sandwich-no-gluten-major))
31
32          :effect (and
33              (not (at_kitchen_bread ?bread-gluten ?bread-no-gluten))
34              (not (at_kitchen_content ?content-gluten ?content-no-gluten))
35              (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
36
37              (at_kitchen_bread ?bread-gluten ?bread-no-gluten-minor)
38              (at_kitchen_content ?content-gluten ?content-no-gluten-minor)
39              (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten-major))
40      )
41
42      (:action make_sandwich_gluten
43          :parameters (?bread-gluten ?bread-no-gluten ?content-gluten ?content-no-gluten
44          ?sandwich-gluten ?sandwich-no-gluten ?bread-gluten-minor ?content-gluten-minor
45          ?sandwich-gluten-major - num)
46
47          :precondition (and
48              (at_kitchen_bread ?bread-gluten ?bread-no-gluten)
49              (at_kitchen_content ?content-gluten ?content-no-gluten)
50              (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
51              (next ?bread-gluten-minor ?bread-gluten)
52              (next ?content-gluten-minor ?content-gluten)
53              (next ?sandwich-gluten ?sandwich-gluten-major)
54          )
55
56          :effect(and
57              (not (at_kitchen_bread ?bread-gluten ?bread-no-gluten))
58              (not (at_kitchen_content ?content-gluten ?content-no-gluten))
59              (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
60              (at_kitchen_bread ?bread-gluten-minor ?bread-no-gluten)
61              (at_kitchen_content ?content-gluten-minor ?content-no-gluten)
62              (at_kitchen_sandwich ?sandwich-gluten-major ?sandwich-no-gluten))
63      )
64
65      (:action make_sandwich_gluten_bread_no_gluten
66          :parameters (?bread-gluten ?bread-no-gluten ?content-gluten ?content-no-gluten
67          ?sandwich-gluten ?sandwich-no-gluten ?bread-no-gluten-minor ?content-gluten-minor
68          ?sandwich-gluten-major - num)
69
70          :precondition (and
71              (at_kitchen_bread ?bread-gluten ?bread-no-gluten)
72              (at_kitchen_content ?content-gluten ?content-no-gluten)
73              (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
74              (next ?bread-no-gluten-minor ?bread-no-gluten)
75              (next ?content-gluten-minor ?content-gluten)
76              (next ?sandwich-gluten ?sandwich-gluten-major))
77
78          :effect (and
79              (not (at_kitchen_bread ?bread-gluten ?bread-no-gluten))
80              (not (at_kitchen_content ?content-gluten ?content-no-gluten))
81              (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
82              (at_kitchen_bread ?bread-gluten ?bread-no-gluten-minor)
```

```
            (at_kitchen_content ?content-gluten-minor ?content-no-gluten)
            (at_kitchen_sandwich ?sandwich-gluten-major ?sandwich-no-gluten))
    )


    (:action make_sandwich_gluten_content_no_gluten
        :parameters (?bread-gluten ?bread-no-gluten ?content-gluten ?content-no-gluten
        ?sandwich-gluten ?sandwich-no-gluten ?bread-no-gluten-minor ?content-gluten-minor
        ?sandwich-gluten-major - num)

        :precondition (and
            (at_kitchen_bread ?bread-gluten ?bread-no-gluten)
            (at_kitchen_content ?content-gluten ?content-no-gluten)
            (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
            (next ?bread-no-gluten-minor ?bread-no-gluten)
            (next ?content-gluten-minor ?content-gluten)
            (next ?sandwich-gluten ?sandwich-gluten-major))

        :effect (and
            (not (at_kitchen_bread ?bread-gluten ?bread-no-gluten))
            (not (at_kitchen_content ?content-gluten ?content-no-gluten))
            (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
            (at_kitchen_bread ?bread-gluten ?bread-no-gluten-minor)
            (at_kitchen_content ?content-gluten-minor ?content-no-gluten)
            (at_kitchen_sandwich ?sandwich-gluten-major ?sandwich-no-gluten))
    )

    (:action put_on_tray
        :parameters (?sandwich-gluten ?sandwich-no-gluten ?sandwich-gluten-2
        ?sandwich-no-gluten-2 ?sandwich-gluten-minor ?sandwich-gluten-2-major - num ?t - tray)

        :precondition (and
            (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
            (ontray ?sandwich-gluten-2 ?sandwich-no-gluten-2 ?t)
            (at ?t kitchen)
            (next ?sandwich-gluten-minor ?sandwich-gluten)
            (next ?sandwich-gluten-2 ?sandwich-gluten-2-major))

        :effect (and
            (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
            (not (ontray ?sandwich-gluten-2 ?sandwich-no-gluten ?t))
            (at_kitchen_sandwich ?sandwich-gluten-minor ?sandwich-no-gluten)
            (ontray ?sandwich-gluten-2-major ?sandwich-no-gluten-2 ?t))
    )



    (:action put_on_tray_no_gluten
        :parameters (?sandwich-gluten ?sandwich-no-gluten ?sandwich-gluten-2
        ?sandwich-no-gluten-2 ?sandwich-no-gluten-minor ?sandwich-no-gluten-2-major - num
        ?t - tray)
        :precondition (and
            (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten)
            (ontray ?sandwich-gluten-2 ?sandwich-no-gluten-2 ?t)
            (at ?t kitchen)
            (next ?sandwich-no-gluten-minor ?sandwich-no-gluten)
            (next ?sandwich-no-gluten-2 ?sandwich-no-gluten-2-major))

        :effect (and
            (not (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten))
            (not (ontray ?sandwich-gluten-2 ?sandwich-no-gluten ?t))
            (at_kitchen_sandwich ?sandwich-gluten ?sandwich-no-gluten-minor)
            (ontray ?sandwich-gluten-2 ?sandwich-no-gluten-2-major ?t))
    )

```

```
148     (:action move_tray
149         :parameters (?t - tray ?p1 ?p2 - place)
150 
151         :precondition (and (at ?t ?p1))
152 
153         :effect (and
154             (not (at ?t ?p1))
155             (at ?t ?p2))
156     )
157 
158     (:action serve_sandwich_no_gluten
159         :parameters (?c - child ?p - place ?t - tray ?sandwich-gluten ?sandwich-no-gluten
160         ?served ?sandwich-no-gluten-minor ?served-major - num)
161 
162         :precondition (and
163             (allergic_gluten ?c)
164             (ontray ?sandwich-gluten ?sandwich-no-gluten ?t)
165             (waiting ?c ?p)
166             (at ?t ?p)
167             (served_childs ?served)
168             (next ?sandwich-no-gluten-minor ?sandwich-no-gluten)
169             (next ?served ?served-major))
170 
171         :effect (and
172             (not (ontray ?sandwich-gluten ?sandwich-no-gluten ?t))
173             (not (waiting ?c ?p))
174             (ontray ?sandwich-gluten ?sandwich-no-gluten-minor ?t)
175             (served_childs ?served-major))
176     )
177 
178     (:action serve_sandwich_gluten
179     :parameters (?c - child ?p - place ?t - tray ?sandwich-gluten ?sandwich-no-gluten
180     ?served ?sandwich-gluten-minor ?served-major - num)
181 
182         :precondition (and
183             (ontray ?sandwich-gluten ?sandwich-no-gluten ?t)
184             (at ?t ?p)
185             (waiting ?c ?p)
186             (not_allergic_gluten ?c)
187             (served_childs ?served)
188             (next ?sandwich-gluten-minor ?sandwich-gluten)
189             (next ?served ?served-major))
190 
191         :effect (and
192             (not (ontray ?sandwich-gluten ?sandwich-no-gluten ?t))
193             (not (waiting ?c ?p))
194             (ontray ?sandwich-gluten-minor ?sandwich-no-gluten ?t)
195             (served_childs ?served-major))
196     )
197 
198 )
```

9.1.5.2 Problema

El problema de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el problema íntegro de esta versión.

```
1  ; child-snack task with 6 children and 0.4 gluten factor
2  ; constant factor of 1.3
3  ; random seed: 234324
4 
5  (define (problem prob-snack)
6    (:domain child-snack)
```

```
7     (:objects
8       number_0 number_1 number_2 number_3 number_4 number_5 number_6 - num
9       child1 child2 child3 child4 child5 child6 - child
10      tray1 tray2 - tray
11      table1 table2 table3 - place
12    )
13    (:init
14      (ontray number_0 number_0 tray2)
15      (ontray number_0 number_0 tray1)
16      (at_kitchen_content number_4 number_2)
17      (at_kitchen_bread number_4 number_2)
18      (at_kitchen_sandwich number_0 number_0)
19      (next number_5 number_6)
20      (next number_4 number_5)
21      (next number_3 number_4)
22      (next number_2 number_3)
23      (next number_1 number_2)
24      (next number_0 number_1)
25      (served_childs number_0)
26       (at tray1 kitchen)
27       (at tray2 kitchen)
28       (allergic_gluten child1)
29       (allergic_gluten child5)
30       (not_allergic_gluten child2)
31       (not_allergic_gluten child3)
32       (not_allergic_gluten child4)
33       (not_allergic_gluten child6)
34       (waiting child1 table2)
35       (waiting child2 table1)
36       (waiting child3 table3)
37       (waiting child4 table2)
38       (waiting child5 table1)
39       (waiting child6 table1)
40    )
41    (:goal
42      (served_childs number_6)
43    )
44  )
```

9.1.6 Versión 08

9.1.6.1 Dominio

El dominio de esta versión varía de manera notable respecto al de cualquiera de las versiones anteriores, por lo que se va a incluir el dominio íntegro de esta versión.

```
1  (define (domain child-snack)
2
3      (:requirements :typing :equality)
4      (:types
5      object
6      child - object
7      tray - object
8      place - object)
9      (:constants kitchen - place)
10
11     (:predicates
12         (allergic_gluten ?c - object)
13         (not_allergic_gluten ?c - object)
14         (served ?c - object)
15         (at ?t - object ?p - object)
16         (waiting ?c - object ?p - object))
17
```

```
18      (:functions
19          (Nbread-portion-at_kitchen_bread)
20          (Nbread-portion-at_kitchen_bread_no_gluten_bread)
21          (Ncontent-portion-at_kitchen_content)
22          (Ncontent-portion-at_kitchen_content_no_gluten_content)
23          (Nsandwich-at_kitchen_sandwich)
24          (Nsandwich-at_kitchen_sandwich_no_gluten_sandwich)
25          (Nsandwich-ontray ?tray - object)
26          (Nsandwich-ontray_no_gluten_sandwich ?tray - object))
27
28      (:action make_sandwich_no_gluten
29          :parameters ()
30
31          :precondition (and
32              (>= (Nbread-portion-at_kitchen_bread_no_gluten_bread) 1)
33              (>= (Ncontent-portion-at_kitchen_content_no_gluten_content) 1))
34
35          :effect (and
36              (increase (Nsandwich-at_kitchen_sandwich_no_gluten_sandwich) 1)
37              (decrease (Nbread-portion-at_kitchen_bread_no_gluten_bread) 1)
38              (increase (Ncontent-portion-at_kitchen_content_no_gluten_content) 1))
39      )
40
41      (:action make_sandwich_gluten
42          :parameters ()
43
44          :precondition (and
45              (>= (Nbread-portion-at_kitchen_bread) 1)
46              (>= (Ncontent-portion-at_kitchen_content) 1))
47
48          :effect (and
49              (increase (Nsandwich-at_kitchen_sandwich) 1)
50              (decrease (Nbread-portion-at_kitchen_bread) 1)
51              (increase (Ncontent-portion-at_kitchen_content) 1))
52      )
53
54      (:action make_sandwich_gluten_bread_no_gluten
55          :parameters ()
56
57          :precondition (and
58              (>= (Nbread-portion-at_kitchen_bread_no_gluten_bread) 1)
59              (>= (Ncontent-portion-at_kitchen_content) 1))
60
61          :effect (and
62              (increase (Nsandwich-at_kitchen_sandwich) 1)
63              (decrease (Nbread-portion-at_kitchen_bread_no_gluten_bread) 1)
64              (increase (Ncontent-portion-at_kitchen_content) 1))
65      )
66
67      (:action make_sandwich_gluten_content_no_gluten
68          :parameters ()
69
70          :precondition (and
71              (>= (Nbread-portion-at_kitchen_bread) 1)
72              (>= (Ncontent-portion-at_kitchen_content_no_gluten_content) 1))
73
74          :effect (and
75              (increase (Nsandwich-at_kitchen_sandwich) 1)
76              (decrease (Nbread-portion-at_kitchen_bread) 1)
77              (increase (Ncontent-portion-at_kitchen_content_no_gluten_content) 1))
78      )
79
80
81      (:action put_on_tray_gluten
82          :parameters (?t - tray)
```

```
83
84          :precondition (and
85              (>= (Nsandwich-at_kitchen_sandwich) 1)
86              (at ?t kitchen))
87
88          :effect (and
89              (increase (Nsandwich-ontray ?t) 1)
90              (decrease (Nsandwich-at_kitchen_sandwich) 1))
91      )
92
93
94      (:action put_on_tray_no_gluten
95          :parameters (?t - tray)
96
97          :precondition (and
98              (>= (Nsandwich-at_kitchen_sandwich_no_gluten_sandwich) 1)
99              (at ?t kitchen))
100
101         :effect (and
102             (increase (Nsandwich-ontray_no_gluten_sandwich ?t) 1)
103             (decrease (Nsandwich-at_kitchen_sandwich_no_gluten_sandwich) 1))
104     )
105
106
107     (:action serve_sandwich_no_gluten
108         :parameters (?t - tray ?c - child ?p - place)
109
110         :precondition (and
111             (at ?t ?p)
112             (>= (Nsandwich-ontray_no_gluten_sandwich ?t) 1)
113             (waiting ?c ?p))
114
115         :effect (and
116             (not (waiting ?c ?p))
117             (served ?c)
118             (decrease (Nsandwich-ontray_no_gluten_sandwich ?t) 1))
119     )
120
121
122     (:action serve_sandwich_gluten
123         :parameters (?t - tray ?c - child ?p - place)
124
125         :precondition (and
126             (at ?t ?p)
127             (>= (Nsandwich-ontray ?t) 1)
128             (waiting ?c ?p)
129             (not_allergic_gluten ?c))
130
131         :effect (and
132             (not (waiting ?c ?p))
133             (served ?c)
134             (decrease (Nsandwich-ontray ?t) 1))
135     )
136
137
138     (:action move_tray
139     :parameters (?t - tray ?p1 ?p2 - place)
140
141     :precondition (and (at ?t ?p1))
142
143     :effect (and
144         (not (at ?t ?p1))
145         (at ?t ?p2))
146     )
147 )
```

9.1.6.2 Problema

El problema de esta versión varía de manera notable respecto al de cualquiera de las versiones anteriores, por lo que se va a incluir el probema íntegro de esta versión.

```
(define (problem prob-snack)
    (:domain child-snack)
    (:objects
    kitchen table1 table2 table3 - place
    child1 child2 child3 child4 child5 child6 - child
    tray1 tray2 - tray
)
    (:init
    (at tray1 kitchen)
    (at tray2 kitchen)
    (allergic_gluten child1)
    (allergic_gluten child5)
    (not_allergic_gluten child2)
    (not_allergic_gluten child3)
    (not_allergic_gluten child4)
    (not_allergic_gluten child6)
    (waiting child1 table2)
    (waiting child2 table1)
    (waiting child3 table3)
    (waiting child4 table2)
    (waiting child5 table1)
    (waiting child6 table1)
    ;;(= (notexist ) 8)
    (= (Nsandwich-ontray tray1) 0)
    (= (Nsandwich-ontray tray2) 0)
    (= (Nsandwich-at_kitchen_sandwich_no_gluten_sandwich ) 0)
    (= (Nbread-portion-at_kitchen_bread_no_gluten_bread ) 2)
    (= (Ncontent-portion-at_kitchen_content_no_gluten_content ) 2)
    (= (Nbread-portion-at_kitchen_bread ) 4)
    (= (Ncontent-portion-at_kitchen_content ) 4)
    (= (Nsandwich-at_kitchen_sandwich ) 0)
    (= (Nsandwich-ontray_no_gluten_sandwich tray1) 0)
    (= (Nsandwich-ontray_no_gluten_sandwich tray2) 0)
)
    (:goal (and
    (served child1)
    (served child2)
    (served child3)
    (served child4)
    (served child5)
    (served child6)
)))
```

La definición de los problemas de esta versión es idéntica a la definición de la versión numérica con dígitos generada automáticamente.

9.1.7 Versión 09

9.1.7.1 Dominio

Se elimina el predicado `(waiting ?c ?p)` y en su lugar se indica el lugar del niño cambiando los predicados `(allergic_gluten ?c)`, `(not_allergic_gluten ?c)` y `(waiting ?c ?p)` por `((allergic_gluten ?c ?p))` y `((not_allergic_gluten`

?c ?p)) .

```
(:action serve_sandwich_no_gluten
    :parameters (?s - sandwich ?c - child ?t - tray ?p - place)

    :precondition (and
        (allergic_gluten ?c ?p)
        (ontray ?s ?t)
        (not (served ?c))
        (no_gluten_sandwich ?s)
        (at ?t ?p))

    :effect (and
        (not (ontray ?s ?t))
        (served ?c))
)
```

```
(:action serve_sandwich
    :parameters (?s - sandwich ?c - child ?t - tray ?p - place)

    :precondition (and
        (not_allergic_gluten ?c ?p)
        (not (served ?c))
        (ontray ?s ?t)
        (at ?t ?p))

    :effect (and
        (not (ontray ?s ?t))
        (served ?c))
)
```

9.1.7.2 Problema

Se aplica el mismo cambio sobre el predicado `(waiting ?c ?p)` que se indica en la sección de dominio.

```
(allergic_gluten child1 table2)
(allergic_gluten child5 table1)
(not_allergic_gluten child2 table1)
(not_allergic_gluten child3 table3)
(not_allergic_gluten child4 table2)
(not_allergic_gluten child6 table1)
```

9.1.8 Versión 11

9.1.8.1 Dominio

En este dominio, cambian el predicado `(not_allergic_gluten ?c)` por `(not(allergic_gluten ?c)` y el predicado `(served ?c)` por `(not(waiting ?c ?p))` .

```
(:action serve_sandwich_no_gluten
    :parameters (?s - sandwich ?c - child ?t - tray ?p - place)

    :precondition (and
        (allergic_gluten ?c)
        (ontray ?s ?t)
        (waiting ?c ?p)
        (no_gluten_sandwich ?s)
        (at ?t ?p))

    :effect (and
        (not (ontray ?s ?t))
        (not(waiting ?c ?p)))
)
```

```
(:action serve_sandwich
    :parameters (?s - sandwich ?c - child ?t - tray ?p - place)
    :precondition (and (not(allergic_gluten ?c))
       (waiting ?c ?p)
    (ontray ?s ?t)
    (at ?t ?p))
    :effect (and (not (ontray ?s ?t))
    (not(waiting ?c ?p)))
)
```

9.1.8.2 Problema

En el problema, cambian los predicados de los objetivos de `(served ?c)` a `(not (waiting ?c ?p))` .

```
(not (waiting child1 table2))
(not (waiting child2 table1))
(not (waiting child3 table3))
(not (waiting child4 table2))
(not (waiting child5 table1))
(not (waiting child6 table1))
```

9.1.9 Versión 12

9.1.9.1 Dominio

El dominio de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el dominio íntegro de esta versión.

```
1  (define (domain child-snack)
2      (:requirements :typing :equality)
3      (:types child bread-portion content-portion sandwich tray place num)
4      (:constants kitchen - place)
5
6      (:predicates
7          (at_kitchen_bread ?b - bread-portion)
8          (at_kitchen_content ?c - content-portion)
9          (at_kitchen_sandwich ?s - sandwich)
10         (no_gluten_bread ?b - bread-portion)
11         (no_gluten_content ?c - content-portion)
12         (ontray ?s - sandwich ?t - tray)
13         (no_gluten_sandwich ?s - sandwich)
```

```
14          (allergic_gluten ?ch - child)
15          (not_allergic_gluten ?ch - child)
16          (served ?ch - child)
17          (waiting ?ch - child ?p - place)
18          (at ?t - tray ?p - place)
19          (notexist ?s - sandwich)
20          (num_sandwich_no_gluten ?n - num)
21          (num_sandwich_gluten ?n - num)
22          (num_sandwich_ontray ?n - num)
23          (next ?minor - num ?major - num)
24          (not_trays_kitchen)
25          (first_load))
26
27      (:action make_sandwich_no_gluten
28          :parameters (?b - bread-portion ?c - content-portion ?n ?minor - num ?s - sandwich)
29
30          :precondition (and
31              (at_kitchen_bread ?b)
32              (at_kitchen_content ?c)
33              (no_gluten_bread ?b)
34              (no_gluten_content ?c)
35              (num_sandwich_no_gluten ?n)
36              (next ?minor ?n)
37              (notexist ?s))
38
39           :effect (and
40              (not (at_kitchen_bread ?b))
41              (not (at_kitchen_content ?c))
42              (at_kitchen_sandwich ?s)
43              (no_gluten_sandwich ?s)
44              (not (num_sandwich_no_gluten ?n))
45              (num_sandwich_no_gluten ?minor)
46              (not (notexist ?s)))
47      )
48
49      (:action make_sandwich
50          :parameters (?b - bread-portion ?c - content-portion ?n ?minor - num ?s - sandwich)
51
52          :precondition (and
53              (at_kitchen_bread ?b)
54              (at_kitchen_content ?c)
55              (num_sandwich_no_gluten number_0)
56              (num_sandwich_gluten ?n)
57              (next ?minor ?n)
58              (notexist ?s))
59
60          :effect (and
61              (not (at_kitchen_bread ?b))
62              (not (at_kitchen_content ?c))
63              (at_kitchen_sandwich ?s)
64              (not (num_sandwich_gluten ?n))
65              (num_sandwich_gluten ?minor)
66              (not (notexist ?s)))
67      )
68
69      (:action move_tray_kitchen
70          :parameters (?t - tray ?p- place)
71
72          :precondition (and
73              (at ?t ?p)
74              (not_trays_kitchen))
75
76          :effect (and
77              (not (at ?t ?p))
78              (at ?t kitchen)
```

```
79              (not (not_trays_kitchen)))
80      )
81
82
83      (:action put_on_tray_first_load
84          :parameters (?s - sandwich ?t - tray ?n ?minor - num)
85
86          :precondition (and
87              (at_kitchen_sandwich ?s)
88              (at ?t kitchen)
89              (not (not_trays_kitchen))
90              (num_sandwich_ontray ?n)
91              (next ?minor ?n)
92              (first_load))
93
94          :effect (and
95              (not (at_kitchen_sandwich ?s))
96              (not (num_sandwich_ontray ?n))
97              (num_sandwich_ontray ?minor)
98              (ontray ?s ?t)
99              (not(first_load)))
100     )
101
102     (:action put_on_tray
103         :parameters (?s ?s2 - sandwich ?t - tray ?n ?minor - num)
104
105         :precondition (and
106             (at_kitchen_sandwich ?s)
107             (at ?t kitchen)
108             (not (first_load))
109             (num_sandwich_ontray ?n)
110             (next ?minor ?n)
111             (ontray ?s2 ?t))
112
113         :effect (and
114             (not (at_kitchen_sandwich ?s))
115             (not (num_sandwich_ontray ?n))
116             (num_sandwich_ontray ?minor)
117             (ontray ?s ?t))
118     )
119
120     ;Restrict the movement of the tray in case there exist the possibility to
121     ;serve a not allergic child.
122     (:action move_tray
123         :parameters (?t - tray ?p1 ?p2 - place ?ch - child ?s - sandwich)
124
125         :precondition (and
126             (at ?t ?p1)
127             (num_sandwich_ontray number_0)
128             (not(and (not_allergic_gluten ?ch)
129             (waiting ?ch ?p1)
130             (ontray ?s ?t))))
131
132         :effect (and
133             (not (at ?t ?p1))
134             (at ?t ?p2))
135     )
136
137     (:action move_tray_no_gluten
138         :parameters (?t - tray ?p1 ?p2 - place ?s - sandwich ?ch - child)
139
140         :precondition (and
141             (at ?t ?p1)
142             (num_sandwich_ontray number_0)
143             (not(and (no_gluten_sandwich ?s)
```

```
144             (allergic_gluten ?ch)
145             (waiting ?ch ?p1)
146             (ontray ?s ?t))))
147
148         :effect (and
149             (not (at ?t ?p1))
150             (at ?t ?p2))
151     )
152
153     (:action serve_sandwich
154
155         :parameters (?ch - child ?p - place ?s - sandwich ?t - tray)
156
157         :precondition (and
158             (not_allergic_gluten ?ch)
159             (waiting ?ch ?p)
160             (ontray ?s ?t)
161             (at ?t ?p))
162
163         :effect (and
164             (not (ontray ?s ?t))
165             (served ?ch))
166     )
167
168     (:action serve_sandwich_no_gluten
169         :parameters (?s - sandwich ?t - tray ?ch - child ?p - place)
170
171         :precondition (and
172             (allergic_gluten ?ch)
173             (ontray ?s ?t)
174             (waiting ?ch ?p)
175             (no_gluten_sandwich ?s)
176             (at ?t ?p))
177
178         :effect (and
179             (not (ontray ?s ?t))
180             (served ?ch))
181     )
182
183 )
```

9.1.9.2 Problema

El problema de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el problema íntegro de esta versión.

```
1  ; child-snack task with 6 children and 0.4 gluten factor
2  ; constant factor of 1.3
3  ; random seed: 234324
4
5  (define (problem prob-snack)
6    (:domain child-snack)
7    (:objects
8      child1 child2 child3 child4 child5 child6 - child
9      bread1 bread2 bread3 bread4 bread5 bread6 - bread-portion
10     content1 content2 content3 content4 content5 content6 - content-portion
11     tray1 tray2 - tray
12     table1 table2 table3 - place
13     sandw1 sandw2 sandw3 sandw4 sandw5 sandw6 - sandwich
14     number_0 number_1 number_2 number_3 number_4 number_5 number_6 - num
15   )
16   (:init
17      (at tray1 kitchen)
```

```
18        (at tray2 kitchen)
19        (at_kitchen_bread bread1)
20        (at_kitchen_bread bread2)
21        (at_kitchen_bread bread3)
22        (at_kitchen_bread bread4)
23        (at_kitchen_bread bread5)
24        (at_kitchen_bread bread6)
25        (at_kitchen_content content1)
26        (at_kitchen_content content2)
27        (at_kitchen_content content3)
28        (at_kitchen_content content4)
29        (at_kitchen_content content5)
30        (at_kitchen_content content6)
31        (no_gluten_bread bread2)
32        (no_gluten_bread bread5)
33        (no_gluten_content content3)
34        (no_gluten_content content6)
35        (allergic_gluten child1)
36        (allergic_gluten child5)
37        (not_allergic_gluten child2)
38        (not_allergic_gluten child3)
39        (not_allergic_gluten child4)
40        (not_allergic_gluten child6)
41        (waiting child1 table2)
42        (waiting child2 table1)
43        (waiting child3 table3)
44        (waiting child4 table2)
45        (waiting child5 table1)
46        (waiting child6 table1)
47        (num_sandwich_no_gluten number_2)
48        (num_sandwich_gluten number_4)
49        (num_sandwich_ontray number_6)
50        (notexist sandw1)
51        (notexist sandw2)
52        (notexist sandw3)
53        (notexist sandw4)
54        (notexist sandw5)
55        (notexist sandw6)
56        (next number_0 number_1)
57        (next number_1 number_2)
58        (next number_2 number_3)
59        (next number_3 number_4)
60        (next number_4 number_5)
61        (next number_5 number_6)
62        (first_load)
63    )
64    (:goal
65      (and
66        (served child1)
67        (served child2)
68        (served child3)
69        (served child4)
70        (served child5)
71        (served child6)
72      )
73    )
74  )
```

9.1.10 Versión automática numérica con palabras

9.1.10.1 Dominio

El dominio de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el dominio íntegro de esta versión.

```
(define (domain child-snack)
    (:requirements :equality :typing)
    (:types child bread-portion content-portion sandwich tray place bread-portion-property1
    bread-portion-property2 bread-portion-num content-portion-property1
    content-portion-property2 content-portion-num sandwich-property1 sandwich-property2
    sandwich-num - object )
    (:constants at_kitchen_bread-true - bread-portion-property1 no_gluten_bread-true
    - bread-portion-property2 bread-portion-property1-none - bread-portion-property1
    at_kitchen_content-true - content-portion-property1 no_gluten_content-true -
    content-portion-property2 content-portion-property1-none - content-portion-property1
    notexist-true - sandwich-property1 notexist-true2 -sandwich-property2
    no_gluten_sandwich-true - sandwich-property1 at_kitchen_content-true -
    content-portion-property1 at_kitchen_sandwich-true - object sandwich-property1-none -
    sandwich-property1 kitchen - place sandwich-property2-none - sandwich-property2)

    (:predicates
        (allergic_gluten ?c - child)
        (served ?c - child)
        (waiting ?c - child ?p - place)
        (at ?t - tray ?p - place)
        (bread-portion-less ?l - bread-portion-num ?m - bread-portion-num)
        (count-bread-portion ?bread-portion - bread-portion ?bread-portion-property1 -
        bread-portion-property1 ?bread-portion-property2 - bread-portion-property2
        ?bread-portion-num - bread-portion-num)
        (content-portion-less ?l - content-portion-num ?m - content-portion-num)
        (count-content-portion ?content-portion - content-portion ?content-portion-property1
        - content-portion-property1 ?content-portion-property2 - content-portion-property2
        ?content-portion-num - content-portion-num)
        (sandwich-less ?l - sandwich-num ?m - sandwich-num)
        (count-sandwich ?sandwich - sandwich ?sandwich-property1 - sandwich-property1
        ?sandwich-property2 - object ?sandwich-num - sandwich-num)
        (bread-portion-lte-sum ?sum1 - bread-portion-num ?sum2 - bread-portion-num ?lte -
        bread-portion-num)
        (bread-portion-bag-size ?bag - bread-portion ?size - bread-portion-num)
        (content-portion-lte-sum ?sum1 - content-portion-num ?sum2 - content-portion-num
        ?lte - content-portion-num)
        (content-portion-bag-size ?bag - content-portion ?size - content-portion-num)
        (sandwich-lte-sum ?sum1 - sandwich-num ?sum2 - sandwich-num ?lte - sandwich-num)
        (sandwich-bag-size ?bag - sandwich ?size - sandwich-num))

    (:action make_sandwich_no_gluten
        :parameters (?s - sandwich ?b - bread-portion ?c - content-portion ?n1 -
        bread-portion-num ?n0 - bread-portion-num ?n2 - bread-portion-num ?n3 -
        bread-portion-num ?n4 - content-portion-num ?n5 - content-portion-num ?n6 -
        content-portion-num ?n7 - content-portion-num ?n8 - sandwich-num ?n9 - sandwich-num
        ?n10 - sandwich-num ?n11 - sandwich-num ?b-size - bread-portion-num ?c-size -
        content-portion-num ?s-size - sandwich-num)

        :precondition (and
            (count-bread-portion ?b at_kitchen_bread-true no_gluten_bread-true ?n1)
            (bread-portion-less ?n0 ?n1)
            (count-bread-portion ?b bread-portion-property1-none no_gluten_bread-true ?n2)
            (bread-portion-less ?n2 ?n3)
            (count-content-portion ?c at_kitchen_content-true no_gluten_content-true ?n4)
            (content-portion-less ?n5 ?n4)
            (count-content-portion ?c content-portion-property1-none no_gluten_content-true
            ?n6)
```

```
58              (content-portion-less ?n6 ?n7)
59              (count-sandwich ?s notexist-true notexist-true2 ?n8)
60              (sandwich-less ?n9 ?n8)
61              (count-sandwich ?s no_gluten_sandwich-true at_kitchen_sandwich-true ?n10)
62              (sandwich-less ?n10 ?n11)
63              (bread-portion-lte-sum ?n1 ?n2 ?b-size)
64              (bread-portion-bag-size ?b ?b-size)
65              (content-portion-lte-sum ?n4 ?n6 ?c-size)
66              (content-portion-bag-size ?c ?c-size)
67              (sandwich-lte-sum ?n8 ?n10 ?s-size)
68              (sandwich-bag-size ?s ?s-size))
69
70          :effect (and
71              (count-bread-portion ?b at_kitchen_bread-true no_gluten_bread-true ?n0)
72              (not (count-bread-portion ?b at_kitchen_bread-true no_gluten_bread-true ?n1))
73              (count-bread-portion ?b bread-portion-property1-none no_gluten_bread-true
74              ?n3)
75              (not (count-bread-portion ?b bread-portion-property1-none no_gluten_bread-true
76              ?n2))
77              (count-content-portion ?c at_kitchen_content-true no_gluten_content-true ?n5)
78              (not (count-content-portion ?c at_kitchen_content-true no_gluten_content-true
79              ?n4))
80              (count-content-portion ?c content-portion-property1-none no_gluten_content-true
81              ?n7)
82              (not (count-content-portion ?c content-portion-property1-none no_gluten_content-true
83              ?n6))
84              (count-sandwich ?s notexist-true notexist-true2 ?n9)
85              (not (count-sandwich ?s notexist-true notexist-true2 ?n8))
86              (count-sandwich ?s no_gluten_sandwich-true at_kitchen_sandwich-true ?n11)
87              (not (count-sandwich ?s no_gluten_sandwich-true at_kitchen_sandwich-true ?n10)))
88      )
89
90      (:action make_sandwich
91          :parameters (?s - sandwich ?b - bread-portion ?c - content-portion ?dc -
92          bread-portion-property2 ?n1 - bread-portion-num ?n0 - bread-portion-num ?n2 -
93          bread-portion-num ?n3 - bread-portion-num ?dc2 - content-portion-property2 ?n4 -
94          content-portion-num ?n5 - content-portion-num ?n6 - content-portion-num ?n7 -
95          content-portion-num ?n8 - sandwich-num ?n9 - sandwich-num ?n10 - sandwich-num ?n11 -
96          sandwich-num ?b-size - bread-portion-num ?c-size - content-portion-num ?s-size -
97          sandwich-num)
98
99          :precondition (and
100             (count-bread-portion ?b at_kitchen_bread-true ?dc ?n1)
101             (bread-portion-less ?n0 ?n1)
102             (count-bread-portion ?b bread-portion-property1-none ?dc ?n2)
103             (bread-portion-less ?n2 ?n3)
104             (count-content-portion ?c at_kitchen_content-true ?dc2 ?n4)
105             (content-portion-less ?n5 ?n4)
106             (count-content-portion ?c content-portion-property1-none ?dc2 ?n6)
107             (content-portion-less ?n6 ?n7)
108             (count-sandwich ?s notexist-true notexist-true2 ?n8)
109             (sandwich-less ?n9 ?n8)
110             (count-sandwich ?s sandwich-property1-none at_kitchen_sandwich-true ?n10)
111             (sandwich-less ?n10 ?n11)
112             (bread-portion-lte-sum ?n1 ?n2 ?b-size)
113             (bread-portion-bag-size ?b ?b-size)
114             (content-portion-lte-sum ?n4 ?n6 ?c-size)
115             (content-portion-bag-size ?c ?c-size)
116             (sandwich-lte-sum ?n8 ?n10 ?s-size)
117             (sandwich-bag-size ?s ?s-size))
118
119         :effect (and
120             (count-bread-portion ?b at_kitchen_bread-true ?dc ?n0)
121             (not (count-bread-portion ?b at_kitchen_bread-true ?dc ?n1))
122             (count-bread-portion ?b bread-portion-property1-none ?dc ?n3)
```

```
123             (not (count-bread-portion ?b bread-portion-property1-none ?dc ?n2))
124             (count-content-portion ?c at_kitchen_content-true ?dc2 ?n5)
125             (not (count-content-portion ?c at_kitchen_content-true ?dc2 ?n4))
126             (count-content-portion ?c content-portion-property1-none ?dc2 ?n7)
127             (not (count-content-portion ?c content-portion-property1-none ?dc2 ?n6))
128             (count-sandwich ?s notexist-true notexist-true2 ?n9)
129             (not (count-sandwich ?s notexist-true notexist-true2 ?n8))
130             (count-sandwich ?s sandwich-property1-none at_kitchen_sandwich-true ?n11)
131             (not (count-sandwich ?s sandwich-property1-none at_kitchen_sandwich-true ?n10)))
132     )
133
134     (:action put_on_tray
135         :parameters (?s - sandwich ?t - tray ?dc - sandwich-property1 ?n1 - sandwich-num ?n0 -
136         sandwich-num ?n2 - sandwich-num ?n3 - sandwich-num ?s-size - sandwich-num)
137
138         :precondition (and
139             (at ?t kitchen)
140             (count-sandwich ?s ?dc at_kitchen_sandwich-true ?n1)
141             (sandwich-less ?n0 ?n1)
142             (count-sandwich ?s ?dc ?t ?n2)
143             (sandwich-less ?n2 ?n3)
144             (sandwich-lte-sum ?n1 ?n2 ?s-size)
145             (sandwich-bag-size ?s ?s-size))
146
147         :effect (and
148             (count-sandwich ?s ?dc at_kitchen_sandwich-true ?n0)
149             (not (count-sandwich ?s ?dc at_kitchen_sandwich-true ?n1))
150             (count-sandwich ?s ?dc ?t ?n3)
151             (not (count-sandwich ?s ?dc ?t ?n2)))
152     )
153
154     (:action move_tray_serve_sandwich
155         :parameters (?t - tray ?p1 - place ?p2 - place ?s - sandwich ?c - child ?dc -
156         sandwich-property1 ?n1 - sandwich-num ?n0 - sandwich-num ?n2 - sandwich-num ?n3 -
157         sandwich-num ?s-size - sandwich-num)
158
159         :precondition (and
160             (at ?t ?p1)
161             (waiting ?c ?p2)
162             (count-sandwich ?s ?dc ?t ?n1)
163             (sandwich-less ?n0 ?n1)
164             (count-sandwich ?s ?dc sandwich-property2-none ?n2)
165             (sandwich-less ?n2 ?n3)
166             (sandwich-lte-sum ?n1 ?n2 ?s-size)
167             (sandwich-bag-size ?s ?s-size))
168
169         :effect (and
170             (not (at ?t ?p1))
171             (at ?t ?p2)
172             (served ?c)
173             (count-sandwich ?s ?dc ?t ?n0)
174             (not (count-sandwich ?s ?dc ?t ?n1))
175             (count-sandwich ?s ?dc sandwich-property2-none ?n3)
176             (not (count-sandwich ?s ?dc sandwich-property2-none ?n2)))
177     )
178
179     (:action move_tray_serve_sandwich_no_gluten
180         :parameters (?t - tray ?p1 - place ?p2 - place ?s - sandwich ?c - child ?n1 -
181         sandwich-num ?n0 - sandwich-num ?n2 - sandwich-num ?n3 - sandwich-num ?s-size -
182         sandwich-num)
183
184         :precondition (and
185             (at ?t ?p1)
186             (waiting ?c ?p2)
187             (allergic_gluten ?c)
```

```
188             (count-sandwich ?s no_gluten_sandwich-true ?t ?n1)
189             (sandwich-less ?n0 ?n1)
190             (count-sandwich ?s no_gluten_sandwich-true sandwich-property2-none ?n2)
191             (sandwich-less ?n2 ?n3)
192             (sandwich-lte-sum ?n1 ?n2 ?s-size)
193             (sandwich-bag-size ?s ?s-size))
194
195         :effect (and
196             (not (at ?t ?p1))
197             (at ?t ?p2)
198             (served ?c)
199             (count-sandwich ?s no_gluten_sandwich-true ?t ?n0)
200             (not (count-sandwich ?s no_gluten_sandwich-true ?t ?n1))
201             (count-sandwich ?s no_gluten_sandwich-true sandwich-property2-none ?n3)
202             (not (count-sandwich ?s no_gluten_sandwich-true sandwich-property2-none ?n2)))
203     )
204
205     (:action move_tray_kitchen
206         :parameters (?t - tray ?p1 - place)
207
208         :precondition (at ?t ?p1)
209
210         :effect (and
211             (not (at ?t ?p1))
212             (at ?t kitchen))
213     )
214
215 )
```

9.1.10.2 Problema

El problema de esta versión varía de manera notable respecto al de cualquiera de las otras versiones, por lo que se va a incluir el problema íntegro de esta versión.

```
1  (define (problem prob-snack)
2  (:domain child-snack)
3  (:objects
4  tray1 tray2  - tray
5  sandwich-property1-none notexist-true no_gluten_sandwich-true  - sandwich-property1
6  bread-portion-property1-none at_kitchen_bread-true  - bread-portion-property1
7  content-portion-property2-none no_gluten_content-true  - content-portion-property2
8  bread-portion-bag1 bread-portion-bag2  - bread-portion
9  bread-portion-property2-none no_gluten_bread-true  - bread-portion-property2
10 sandwich-bag1  - sandwich
11 sandwich-num1 sandwich-num2 sandwich-num3 sandwich-num4 sandwich-num5 sandwich-num6
12 sandwich-num7 sandwich-num8 sandwich-num0  - sandwich-num
13 bread-portion-num1 bread-portion-num2 bread-portion-num3 bread-portion-num4
14 bread-portion-num0  - bread-portion-num
15 content-portion-num1 content-portion-num2 content-portion-num3 content-portion-num4
16 content-portion-num0  - content-portion-num
17 content-portion-bag1 content-portion-bag2  - content-portion
18 sandwich-property2-none at_kitchen_sandwich-true notexist-true2  - sandwich-property2
19 content-portion-property1-none at_kitchen_content-true  - content-portion-property1
20 child1 child2 child3 child4 child5 child6  - child
21 kitchen table1 table2 table3  - place
22 )
23 (:init (at tray2 kitchen)
24 (waiting child2 table1)
25 (at tray1 kitchen)
26 (waiting child1 table2)
27 (waiting child3 table3)
28 (allergic_gluten child1)
29 (waiting child6 table1)
```

```
(allergic_gluten child5)
(waiting child4 table2)
(waiting child5 table1)
(bread-portion-less bread-portion-num1 bread-portion-num2)
(bread-portion-less bread-portion-num2 bread-portion-num3)
(bread-portion-less bread-portion-num3 bread-portion-num4)
(content-portion-less content-portion-num1 content-portion-num2)
(content-portion-less content-portion-num2 content-portion-num3)
(content-portion-less content-portion-num3 content-portion-num4)
(sandwich-less sandwich-num1 sandwich-num2)
(sandwich-less sandwich-num2 sandwich-num3)
(sandwich-less sandwich-num3 sandwich-num4)
(sandwich-less sandwich-num4 sandwich-num5)
(sandwich-less sandwich-num5 sandwich-num6)
(sandwich-less sandwich-num6 sandwich-num7)
(sandwich-less sandwich-num7 sandwich-num8)
(count-bread-portion bread-portion-bag1 at_kitchen_bread-true
bread-portion-property2-none bread-portion-num4)
(count-bread-portion bread-portion-bag2 at_kitchen_bread-true no_gluten_bread-true
bread-portion-num2)
(count-content-portion content-portion-bag1 at_kitchen_content-true
content-portion-property2-none content-portion-num4)
(count-content-portion content-portion-bag2 at_kitchen_content-true no_gluten_content-true
content-portion-num2)
(count-sandwich sandwich-bag1 notexist-true notexist-true2 sandwich-num8)
(bread-portion-less bread-portion-num0 bread-portion-num1)
(content-portion-less content-portion-num0 content-portion-num1)
(sandwich-less sandwich-num0 sandwich-num1)
(count-bread-portion bread-portion-bag1 bread-portion-property1-none
bread-portion-property2-none bread-portion-num0)
(count-bread-portion bread-portion-bag1 at_kitchen_bread-true no_gluten_bread-true
bread-portion-num0)
(count-bread-portion bread-portion-bag1 bread-portion-property1-none no_gluten_bread-true
bread-portion-num0)
(count-bread-portion bread-portion-bag2 bread-portion-property1-none
bread-portion-property2-none bread-portion-num0)
(count-bread-portion bread-portion-bag2 at_kitchen_bread-true
bread-portion-property2-none bread-portion-num0)
(count-bread-portion bread-portion-bag2 bread-portion-property1-none
no_gluten_bread-true bread-portion-num0)
(count-content-portion content-portion-bag1 content-portion-property1-none
content-portion-property2-none content-portion-num0)
(count-content-portion content-portion-bag1 at_kitchen_content-true
no_gluten_content-true content-portion-num0)
(count-content-portion content-portion-bag1 content-portion-property1-none
no_gluten_content-true content-portion-num0)
(count-content-portion content-portion-bag2 content-portion-property1-none
content-portion-property2-none content-portion-num0)
(count-content-portion content-portion-bag2 at_kitchen_content-true
content-portion-property2-none content-portion-num0)
(count-content-portion content-portion-bag2 content-portion-property1-none
no_gluten_content-true content-portion-num0)
(count-sandwich sandwich-bag1 sandwich-property1-none sandwich-property2-none
sandwich-num0)
(count-sandwich sandwich-bag1 notexist-true sandwich-property2-none sandwich-num0)
(count-sandwich sandwich-bag1 no_gluten_sandwich-true sandwich-property2-none
sandwich-num0)
(count-sandwich sandwich-bag1 no_gluten_sandwich-true at_kitchen_sandwich-true
sandwich-num0)
(count-sandwich sandwich-bag1 notexist-true at_kitchen_sandwich-true sandwich-num0)
(count-sandwich sandwich-bag1 sandwich-property1-none at_kitchen_sandwich-true
sandwich-num0)
(count-sandwich sandwich-bag1 sandwich-property1-none notexist-true2 sandwich-num0)
(count-sandwich sandwich-bag1 no_gluten_sandwich-true notexist-true2 sandwich-num0)
(count-sandwich sandwich-bag1 no_gluten_sandwich-true tray1 sandwich-num0)
```

```
(count-sandwich sandwich-bag1 notexist-true tray1 sandwich-num0)
(count-sandwich sandwich-bag1 sandwich-property1-none tray1 sandwich-num0)
(count-sandwich sandwich-bag1 sandwich-property1-none tray2 sandwich-num0)
(count-sandwich sandwich-bag1 notexist-true tray2 sandwich-num0)
(count-sandwich sandwich-bag1 no_gluten_sandwich-true tray2 sandwich-num0)
(bread-portion-bag-size bread-portion-bag1 bread-portion-num4)
(bread-portion-bag-size bread-portion-bag2 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num0 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num1 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num2 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num3 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num4 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num0 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num1 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num2 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num3 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num2 bread-portion-num0 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num2 bread-portion-num1 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num2 bread-portion-num2 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num3 bread-portion-num0 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num3 bread-portion-num1 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num4 bread-portion-num0 bread-portion-num4)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num0 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num1 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num0 bread-portion-num2 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num0 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num1 bread-portion-num1 bread-portion-num2)
(bread-portion-lte-sum bread-portion-num2 bread-portion-num0 bread-portion-num2)
(content-portion-bag-size content-portion-bag1 content-portion-num4)
(content-portion-bag-size content-portion-bag2 content-portion-num2)
(content-portion-lte-sum content-portion-num0 content-portion-num0 content-portion-num4)
(content-portion-lte-sum content-portion-num0 content-portion-num1 content-portion-num4)
(content-portion-lte-sum content-portion-num0 content-portion-num2 content-portion-num4)
(content-portion-lte-sum content-portion-num0 content-portion-num3 content-portion-num4)
(content-portion-lte-sum content-portion-num0 content-portion-num4 content-portion-num4)
(content-portion-lte-sum content-portion-num1 content-portion-num0 content-portion-num4)
(content-portion-lte-sum content-portion-num1 content-portion-num1 content-portion-num4)
(content-portion-lte-sum content-portion-num1 content-portion-num2 content-portion-num4)
(content-portion-lte-sum content-portion-num1 content-portion-num3 content-portion-num4)
(content-portion-lte-sum content-portion-num2 content-portion-num0 content-portion-num4)
(content-portion-lte-sum content-portion-num2 content-portion-num1 content-portion-num4)
(content-portion-lte-sum content-portion-num2 content-portion-num2 content-portion-num4)
(content-portion-lte-sum content-portion-num3 content-portion-num0 content-portion-num4)
(content-portion-lte-sum content-portion-num3 content-portion-num1 content-portion-num4)
(content-portion-lte-sum content-portion-num4 content-portion-num0 content-portion-num4)
(content-portion-lte-sum content-portion-num0 content-portion-num0 content-portion-num2)
(content-portion-lte-sum content-portion-num0 content-portion-num1 content-portion-num2)
(content-portion-lte-sum content-portion-num0 content-portion-num2 content-portion-num2)
(content-portion-lte-sum content-portion-num1 content-portion-num0 content-portion-num2)
(content-portion-lte-sum content-portion-num1 content-portion-num1 content-portion-num2)
(content-portion-lte-sum content-portion-num2 content-portion-num0 content-portion-num2)
(sandwich-bag-size sandwich-bag1 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num0 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num1 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num2 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num3 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num4 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num5 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num6 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num7 sandwich-num8)
(sandwich-lte-sum sandwich-num0 sandwich-num8 sandwich-num8)
(sandwich-lte-sum sandwich-num1 sandwich-num0 sandwich-num8)
(sandwich-lte-sum sandwich-num1 sandwich-num1 sandwich-num8)
(sandwich-lte-sum sandwich-num1 sandwich-num2 sandwich-num8)
(sandwich-lte-sum sandwich-num1 sandwich-num3 sandwich-num8)
```

```
160 (sandwich-lte-sum sandwich-num1 sandwich-num4 sandwich-num8)
161 (sandwich-lte-sum sandwich-num1 sandwich-num5 sandwich-num8)
162 (sandwich-lte-sum sandwich-num1 sandwich-num6 sandwich-num8)
163 (sandwich-lte-sum sandwich-num1 sandwich-num7 sandwich-num8)
164 (sandwich-lte-sum sandwich-num2 sandwich-num0 sandwich-num8)
165 (sandwich-lte-sum sandwich-num2 sandwich-num1 sandwich-num8)
166 (sandwich-lte-sum sandwich-num2 sandwich-num2 sandwich-num8)
167 (sandwich-lte-sum sandwich-num2 sandwich-num3 sandwich-num8)
168 (sandwich-lte-sum sandwich-num2 sandwich-num4 sandwich-num8)
169 (sandwich-lte-sum sandwich-num2 sandwich-num5 sandwich-num8)
170 (sandwich-lte-sum sandwich-num2 sandwich-num6 sandwich-num8)
171 (sandwich-lte-sum sandwich-num3 sandwich-num0 sandwich-num8)
172 (sandwich-lte-sum sandwich-num3 sandwich-num1 sandwich-num8)
173 (sandwich-lte-sum sandwich-num3 sandwich-num2 sandwich-num8)
174 (sandwich-lte-sum sandwich-num3 sandwich-num3 sandwich-num8)
175 (sandwich-lte-sum sandwich-num3 sandwich-num4 sandwich-num8)
176 (sandwich-lte-sum sandwich-num3 sandwich-num5 sandwich-num8)
177 (sandwich-lte-sum sandwich-num4 sandwich-num0 sandwich-num8)
178 (sandwich-lte-sum sandwich-num4 sandwich-num1 sandwich-num8)
179 (sandwich-lte-sum sandwich-num4 sandwich-num2 sandwich-num8)
180 (sandwich-lte-sum sandwich-num4 sandwich-num3 sandwich-num8)
181 (sandwich-lte-sum sandwich-num4 sandwich-num4 sandwich-num8)
182 (sandwich-lte-sum sandwich-num5 sandwich-num0 sandwich-num8)
183 (sandwich-lte-sum sandwich-num5 sandwich-num1 sandwich-num8)
184 (sandwich-lte-sum sandwich-num5 sandwich-num2 sandwich-num8)
185 (sandwich-lte-sum sandwich-num5 sandwich-num3 sandwich-num8)
186 (sandwich-lte-sum sandwich-num6 sandwich-num0 sandwich-num8)
187 (sandwich-lte-sum sandwich-num6 sandwich-num1 sandwich-num8)
188 (sandwich-lte-sum sandwich-num6 sandwich-num2 sandwich-num8)
189 (sandwich-lte-sum sandwich-num7 sandwich-num0 sandwich-num8)
190 (sandwich-lte-sum sandwich-num7 sandwich-num1 sandwich-num8)
191 (sandwich-lte-sum sandwich-num8 sandwich-num0 sandwich-num8)
192 )
193 (:goal (and
194           (served child1)
195           (served child2)
196           (served child3)
197           (served child4)
198           (served child5)
199           (served child6))
200 )
201 )
```

9.1.11 Versión automática numérica con dígitos

9.1.11.1 Dominio

El dominio de esta versión es idéntico al dominio de la versión 08 a excepción de la definición del operador `serve_sandwich` que en este caso se ha dividido en los operadores `serve_sandwich_1_1` y `serve_sandwich_2_1` .

```
(:action serve_sandwich_1_1
    :parameters (?p - place ?c - child ?t - tray)

    :precondition (and
        (at ?t ?p)
        (waiting ?c ?p)
        (not_allergic_gluten ?c)
        (>= (Nsandwich-ontray ?t) 1))

    :effect (and
        (served ?c)
        (decrease (Nsandwich-ontray ?t) 1))
)
```

```
(:action serve_sandwich_2_1
    :parameters (?p - place ?c - child ?t - tray)

    :precondition (and
        (at ?t ?p)
        (waiting ?c ?p)
        (not_allergic_gluten ?c)
        (>= (Nsandwich-ontray_no_gluten_sandwich ?t) 1))

    :effect (and
        (served ?c)
        (decrease (Nsandwich-ontray_no_gluten_sandwich ?t) 1))
)
```

9.1.11.2 Problema

La definición de los problemas de esta versión es idéntica a la definición de la versión 08.

9.2 Anexo II: planificación del proyecto

En esta sección se abordará tanto la planificación temporal del proyecto como el coste económico estimado para el desarrollo de este proyecto.

9.2.1 Planificación temporal

En el siguiente gráfico de Gantt se puede ver cómo se ha distribuído el tiempo necesario para desarrollar este proyecto. Tras este, se desglosa de manera resumida en qué ha consistido el trabajo desarrollado en cada una de las etapas expuestas en este diagrama.

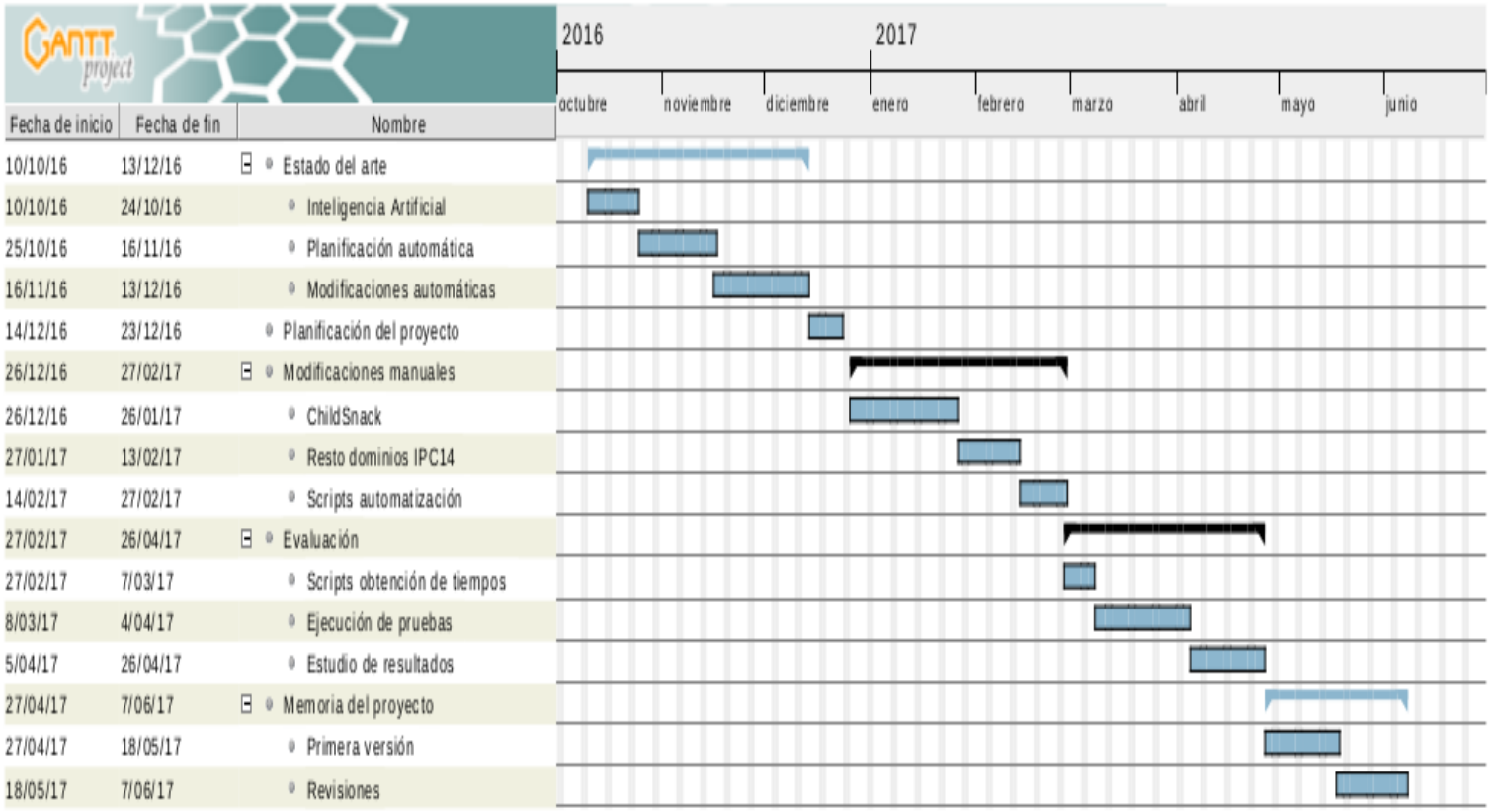

Figura 25: Diagrama de Gantt del proyecto

- **Estado del arte**: el primer paso que se ha dado ha sido investigar sobre lo que se ha hecho en el campo de estudio en el que se va a trabajar. En esta fase la mayor parte de la sección de la memoria que lleva este mismo título ha sido redactada mientras que en otra se ha dejado un pequeño esquema minimalista sobre la materia que sirviera simplemente para saber lo que se ha hecho en este campo para tenerlo en cuenta al implementar la solución. Estas partes que no han sido redactadas para poder ser incluidas directamente en este proyecto han sido luego desarrolladas en la fase de redacción de la memoria del proyecto. Como se puede ver el en órden cronológico de las subtareas, se ha seguido un órden que implique pasar de lo más genérico a cada vez lo más concreto en relación con el tema de este trabajo.
  - **Inteligencia artificial**: como ya se tenían ciertas nociones sobre este área, la búsqueda ha ido encaminada a la parte histórica de la inteligencia artificial y sus diferentes secciones para así incluirlo en la memoria.
  - **Planificación automática**: antes de empezar este proyecto, ya se había trabajado con planificadores automáticos de *PDDL*. Sin embargo, en estos trabajos previos el planificador operaba como una caja negra. Por tanto, esta sección ha sido muy necesaria para así poder interpretar de manera más acertada los resultados obtenidos tras la planificación automática.
  - **Modificaciones automáticas**: como se ha reflejado en la memoria, existe trabajo previo de otros investigadores que han realizado programas para automatizar la representación de dominios en *PDDL*, siendo algunos de ellos independientes o dependientes de dominio. El código de las modificaciones independientes de dominio más prometedoras que se han conseguido ejecutar con éxito han sido incluidas en la sección de pruebas de esta memoria.
- Modificaciones manuales: al estudiar un dominio, es posible encontrar la forma de modificar la representación del mismo para reducir la búsqueda del planificador para así obtener soluciones óptimas y completas en un menor tiempo de búsqueda respecto a la versión original.
  - **ChildSnack**: este es el principal dominio con el que se han realizado todas las pruebas y por ello se ha dedicado más tiempo a este dominio que a la suma del tiempo dedicado a todos los demás dominios analizados.
  - **Resto dominios *IPC*14**: para intentar generalizar algunas de las conclusiones de este proyecto, se han analizado otros dominios de la misma edición en la que se presentó el dominio *ChildSnack* a la *IPC*. Finalmente se han descartado estos resultados debido a la complejidad de la inclusión de estos

resultados con el fin inicial que se tenía de dar una visión más global. Sin embargo, este trabajo al haber sido realizado, se incluye en la planificación real del proyecto.

  - **Scripts automatización**: algunas de las modificaciones realizadas a mano en el dominio de *ChildSnack* se han automatizado mediante scripts para así poder transformar de manera automática los problemas del dominio original.

- **Evaluación**: en esta fase se han evaluado las distintas versiones de *ChildSnack* con varios planificadores automáticos.

  - **Scripts de obtención de tiempos**: se ha creado un script que dado un dominio, una carpeta de problemas y uno de los planificadores para los que está diseñado el dominio, devuelve un archivo de texto plano con todos los tiempos de ejecución, la calidad de la solución así como guarda el plan en caso de haber encontrado la solución al problema.
  - **Ejecución de pruebas**: esta fase ha consistido en la ejecución del script con los distintos parámetros de entrada y comprobación de una muestra de los resultados para verificar que no ha habido ningún problema en la ejecución.
  - **Estudio de resultados**: una vez se ha verificado que los datos obtenidos son correcto, es necesario interpretarlos ya que la comparación de los mismos determinará su exposición en la memoria.

- **Memoria del proyecto**: una vez se han realizado todas las pruebas, se han documentado todo el proceso seguido así como se ha dejado por escrito la interpretación de los resultados obtenidos y las líneas futuras de investigación. Antes de empezar esta fase, únicamente estaba redactada parte de la sección del estado del arte.

  - **Primera versión**: se ha realizado una primera versión de todo el proyecto para entregar a la tutora y que a partir de esta realizase las anotaciones correspondientes de aquello que era necesario modificar.
  - **Revisiones**: las posteriores versiones de la memoria han sido trabajadas tanto a partir de la totalidad de la memoria en distintas revisiones como mediante revisiones concretas de cada una de las secciones de la memoria.

Resumiendo, entre la fecha de inicio y la fecha de finalización del proyecto han trascurrido

$341 dias = \frac{241 dias}{7 dias/semana} = 35 semanas$. Si estimamos una media de $13 horas/semana$ a este proyecto, el total de horas dedicadas al proyecto por el alumno es de $35 semanas * 13 horas/semana = 455 horas$. Estas horas incluyen las horas invertidas en tutorías especificadas a continuación pero no las horas de ejecución de las pruebas automatizadas.

La planificación ha estado fuertemente influida por las diversas reuniones entre el alumno y la tutora. A continuación se incluye esta información que servirá tanto para plasmar la planificación de este proyecto como para la estimación ulterior de la estimación de gasto en recursos humanos para la realización de este proyecto.

1. 21 de septiembre de 2016.
2. 10 de octubre de 2016.
3. 25 de octubre de 2016
4. 23 de noviembre de 2016
5. 14 de diciembre de 2016
6. 20 de febrero de 2017
7. 14 de marzo de 2017
8. 4 de abril de 2017
9. 27 de abril de 2017
10. 24 de mayo de 2017
11. 29 de mayo de 2017

La duración media de estas reuniones ha sido de en torno a 105 minutos, por lo tanto se estima el número de horas invertidas en estas tutorías en
$1.75 horas/tutoria * 11 tutorias = 20 horas$. Se estima a su vez que para 6 de estas tutorías ha sido necesario el trabajo previo de $2 horas$ por parte de la tutora, sumado a $12 horas$ dedicadas para las distintas revisiones de la memoria, lo que hace un total de
$20 + 6 * 2 + 12 = 44 horas$. Esta cifra será más adelante utilizada para estimar el coste económico de recursos humanos.

9.2.2 Estimación de costes económicos

Esta sección versa sobre el coste económico de la realización de este proyecto en caso de haberse tenido que realizar sin contar con ningún material previo (como pueda ser un ordenador de sobremesa) ni con la idiosincracia de pago de personal académico en la universidad. Es decir, la estimación de costes que se muestra toma como marco de referencia la empresa privada así como la separación de trabajadores según las áreas de trabajo que se pueden encontrar en empresas tecnológicas grandes. Según las recomendaciones de Bruselas acatadas por España en el año 2013, se considera una empresa grande a partir de 250 trabajadores[53].

9.2.2.1 Recursos materiales

Tabla 8: Recursos materiales

Concepto	**Unidades**	**Coste/u**	**Total**	**Amortización/mes**	**Amortizado(8meses)**
Ordenador	1	649€	649€	2,08%	107,99€
Pantalla	1	79€	79€	2,08%	13,15€
Router	1	22€	22€	2,08%	3,66€
Total	-	-	750€	-	124,8€

Antes de poder estimar el coste de materiales fungibles, es necesario calcular las horas de electricidad que han sido necesarias para que el ordenador estuviese encendido con el fin de realizar este proyecto. Para ello hay que añadir a las horas de trabajo del alumno frente al ordenador el número de horas de ejecución de las pruebas.

Teniendo en cuenta que todos los artículos y libros citados en la bibliografía han sido consultados a través del ordenador, las únicas horas del proyecto que no se ha utilizado este dispositivo es cuando el alumno se encontraba en una tutoría del proyecto. Por tanto, el número de horas en las que se ha hecho uso del ordenador sin contar las horas de pruebas es de $455horas - 20horas = 435horas$.

Respecto a las horas necesarias para ejecutar las diversas pruebas, primero hay que tomar en cuenta que aunque en la versión que se ha presentado de este trabajo existan 12 dominios distintos de Childsnack, antes de cribar las versiones menos prometedoras el número de estos dominios era de 16 dominios. Estos 4 dominios descartados eran de baja calidad por lo que para esta estimación se asumirá que únicamente resolvían 2 de los 20 problemas en 30 segundos cada uno, por lo que para simplificar la estimación al ser un tiempo nimio esta cantidad no será tomada en cuenta. Teniendo en cuenta que el tiempo límite de búsqueda de los planificadores es de 30 minutos y que dichas pruebas se ejecutaron 2 veces, el tiempo consumido en la prueba de estos

Tabla 9: Coste de materiales fungibles

Concepto	**Unidades**	**Coste/u**	**Total**
Material para escribir	1	9	9
Hora de electricidad	724	0.1358€/KWhora	98,32
Conexión a internet	8	19,2	153,6
Total	-	-	260,92

4 dominios descartados es de $4 * (0.5 horas/problema * 18 problemas)/ejecucion * 2 ejecuciones = 72 horas$. Esta cifra corresponde al número de horas de ejecución necesarias por cada planificador utilizado. Como se han utilizado dos planificadores, el número total será: $72 horas/planificador * 2 planificadores = 144 horas$.

De la versión final se han conservado los tiempos de ejecución para cada planificador, ya que dichos resultados han sido analizados en la sección de evaluación de resultados. Por tanto, teniendo en cuenta que estas pruebas también se han realizado 2 veces, que se han realizado las pruebas con 2 planificadores y tomando la simplificación de descartar el tiempo de las ejecuciones en las que se ha resuelto el problema y añadir un total de $3 horas/ejecucion * 2 ejecuciones = 6 horas$ horas que es un tiempo superior al de la suma del tiempo necesario para la obtención de todas las soluciones para todos los problemas de todas las versiones, obtenemos:

$97 problemasnoresueltosMFF/iteracion * 2 iteraciones +$
$42 problemasnoresueltosLPG - td/iteracion * 2 iteraciones$
$= 278 problemasnoresueltos * 0.5 horas/problemanoresuelto = 139 horas$

Por tanto, el número de horas de electricidad necesarias para alimentar el ordenador durante este proyecto será de $435 + 144 + 6 + 139 horas = 724 horas$ de luz necesarias.

El coste de materiales fungibles es el siguiente:

Por tanto, la estimación de gastos materiales totales para este proyecto es de $124,8 + 260,92 = 385,72$. A esta cantidad habría que añadirle la cantidad no amortizada en los 8 meses de duración de proyecto que sin embargo sin necesarios para comprar el material para desarrollar este proyecto. Es decir, sería necesario disponer a su vez de $750 - 124,8 = 625,2$ de inversión inicial.

Se quiere volver a insistir en que estos datos son estimatorios y simplificados. Otros gastos como el gasto de luz para la iluminación o la calefación durante el invierno han sido descartados.

9.2.2.2 Recursos humanos

Para este proyecto se pueden dividir las tareas realizadas en tantos empleados como la imaginación pueda dar de sí. Estas distintas versiones podrían luego ser estimadas y con ello extraer el coste económico correspondiente a la remuneración de estos trabajadores. Sin embargo, para que esta estimación sea más realista, se va a asumir como coste de desarrollo de este proyecto la composición real que ha tenido (una tutora y un alumno) pero cambiando dichos roles por su equivalente en la empresa privada (como puede ser el project manager y el programador raso).

Tabla 10: Coste recursos humanos

Posición	**Coste/hora**	**Horas**	**Total**
Project manager	35	44	1.540€
Programador	25	455	11.375€
Total	-	-	12.915€

9.2.2.3 Coste total

El coste total de este proyecto será igual al sumatorio de los costes de los materiales fungibles, físicos, el coste en recursos humanos así como el porcentaje de ganancia y el porcentaje de riesgos, como puede ser que el terminal utilizado deje de estar operativo. Este sumatorio se resume en la tabla que se muestra a continuación.

Tabla 11: Coste total del proyecto

Concepto	**Coste**
Recursos físicos	124,8€
Recursos fungibles	260,92€
Recursos humanos	12.915€
Beneficios	2.632,14€
Riesgos	1974,11€
Total	**17.906,97€**

9.3 Anexo III: Regulaciones

En este capítulo se desglosan las regulaciones sociales, legales y económicas de este proyecto.

9.3.1 Regulaciones sociales

Este proyecto está enmarcado dentro del estudio de la planificación automática en *PDDL*. Las conclusiones de este trabajo son útiles principalmente para investigadores de este sector así como también para todo aquel que quiera definir manualmente dominios en *PDDL*, ya que las conclusiones extraídas podrán ser de ayuda para que dicho dominio pueda ser resuelto en el menor tiempo de búsqueda posible.

9.3.2 Regulaciones legales

Para este proyecto la totalidad de herramientas utilizadas han sido de software libre y gratuito. En el caso de los dominios utilizados, haciendo una lectura en diagonal de los artículos de *PDDL* citados se puede ver que la competición *IPC* es una piedra angular en el campo de la investigación de la planificación automática. Para poder testear los dominios presentados a esta competición debido al funcionamiento de *PDDL*, estos necesariamente tienen que ser editables (es decir, la entrada de los problemas no son archivos binarios o de otro tipo que dificulten la edición manual de los mismos). A su vez, la mayoría de los planificadores presentados a esta competición son de software libre y gratuito. Esto permite que se produzca una constante retroalimentación entre investigadores ya que aquellos hallazgos referentes a la planificación son socializados entre los participantes. Esto ha permitido por ejemplo que el planificador de software libre *FF* halla sido presentado por distintos investigadores en distintas versiones del mismo (por ejemplo, *MacroFF* o *Marvin* son implementaciones que parten de *FF*).

Si los planificadores de *PDDL* hubiesen funcionado de otra manera en el que la entrada de los problemas no tuviese que ser necesariamente fácilmente editables y estos dominios no hubiesen sido públicos y accesibles, es muy problable que este proyecto no hubiese existido. Por ello, es imprescindible que tanto esta memoria como los scripts y los dominios utilizados sigan necesariamente esta misma política de compartición de conocimiento para permitir así que otras personas que estén trabajando en la planificación automática puedan sacar el máximo provecho a este trabajo de la misma manera que ha sucedido para la realización de este proyecto. Por tanto, además de encontrar la memoria de este proyecto en formato *PDF* en la biblioteca de la Universidad Carlos III de Madrid, será posible encontrar todo el

código utilizado y la memoria en formato LaTeXen el repositorio personal del alumno[6].

Dicho esto, las licencias de la memoria y del código del proyecto son las siguientes:

- Todos los dominios y scripts creados que pueden encontrarse en el repositorio personal tienen la licencia **GPLv3**[7].
- Esta memoria está sujeta a la licencia de **Creative Commons Reconocimiento-NoComercial-CompartirIgual** 4.0[8].

9.3.3 Regulaciones económicas

Este proyecto puede ayudar a definir mejores dominios en *PDDL*. Sin embargo al no haber desarrollado ningún programa independiente de dominio que realice esta tarea, no es posible obtener rédito económico inmediato de manera directa a partir del trabajo desarrollado.

[6] https://github.com/asdrgil
[7] https://choosealicense.com/licenses/gpl-3.0/
[8] http://creativecommons.org/licenses/by-nc-sa/4.0/

9.4 Anexo IV: Summary of this work

This section includes an English global summary of the whole project.

9.4.1 Abstract

This work is a study that tries to shed light on the possibility of reducing the search time and improve the solution's quality by means of variations of the representation of the domains and problems of automated planning. Hence, an empiric study has been made where several modified versions of domains and problems -some of them generated by hand, others generated by domain independent software from other developers- have been used as the input files for planning.

With this work it is expected to prove the relation between the results of planning and the representation of domains and problems, as well as the importance of the election of the planner, as the results of the same representation with different planners vary. In brief, the suitability of certain domain and problem representations could vary depending on the planner used.

The conclusions obtained from this work are that it confirms the initial hypothesis saying that the variation of the representation of the domains and problems could be the reason of substantial variations in both the search times and the quality of the solutions obtained; these results depend on the planner used. There is still a long path to be travelled regarding the domain independent software focused on modifying the representation of problems and domains as the results of the versions obtained from the execution of these programs where exceeded by some of the manual domains.

Keywords: **automated planning (AP)**, **representatation in AP**, **PDDL**

9.4.2 Introduction

This is an investigation work that tries to find out the correlation in automated planning between different representation forms of the same problem to be solved and the results obtained after the execution of different planners for the different representations.

Within the next sections it can be found the state of the art of this problem, where it will be explained from the most generic section (artificial intelligence) to the most specific sections regarding the different types of domain independent software aimed to modify the representations of both problems and domains in *PDDL*.

Once it has been explained the state of the art of the investigation of this subject, it will be introduced the domain and problems to be modified. This domain will be modified both by hand and using domain independent software. The different versions of the original domain and problems that will be obtained from these modifications will be then used as input for different planners and the result of the planification made by these planners will be collected.

The results obtained will be then compared an analyzed and some conclusions will be raised based on these results. It is known that due to the fact of the limitation of the number of planners used as well as the fact that only one type of problem is being tested, the generalization of the conclusions raised in this document are limited and could vary among different planners and problems. However, as it can be seen in the following sections, these limitations are taken into account in the conclusions obtained and therefore this could just mean that this work could be enlarged with a greater amount of tryouts of different domains, problems and planners.

9.4.3 State of the art

In this section, it will be explained the state of the art of the area where this work is encapsulated. It will be explained from the most generic area (artificial intelligence) to the most concrete, automated planning.

9.4.3.1 Artificial intelligence

This section includes the definition and some historical milestones of artificial intelligence.

9.4.3.1.1 Definition of artificial intelligence

Commonly the Darthmounth conferences of 1956 are considered the germ of artificial intelligence as a research area. These conferences had the objective of studying all the aspects of the intelligence so that it could be described with an accuracy such that a computer could simulate it. The organizer of these conferences, J. McCarthy, coined the term *artificial intelligence* as *"the science and the engineering dedicated to the creation of intelligent machines, specially intelligent computer programs"*.

When the term became popular, several definitions aroused. Stuart J. Russel and Peter Norvig classified them in the following groups:

- Those who defined artificial intelligence as systems that **could think like humans**.
- Those who defined artificial intelligence as systems that **could act like humans**.
- Those who defined artificial intelligence as systems that **could think rationally**.
- Those who defined artificial intelligence as systems that **could act rationally**.

9.4.3.1.2 Evolution of artificial intelligence

Even though the Turing machine was an important preceeding for the development of artificial intelligence, the neural network model of 1943 developed by McCulloch and Pitt it is nowadays considered the first work enclosed in the artificial intelligence field. In this work, they proved that the Turing machine could be implemented using a finite neural network.

The different achievements and phases of artificial intelligence could be summarized as follows:

1957 General Problem Solver is developed. It was designed to be an universal problem solver. It is the main aforegoing of automated planning.

1959 Creation of the simple perceptron.

1972 Creation of the programming language Prolog, which will be later proposed as the native language for the fifth generation machines.

1980-1993 Fifth generation project: project from the government of Japan aimed to lead the next computer revolution.

1997 A supercomputer developed by IBM, Deep Blue, wins a six games match against the chess world champion.

9.4.3.2 Automated planning

This section contains the basic information about automated planning.

9.4.3.2.1 Definition

Succintly, automated planning could be defined as *"the art and practise of thinking before acting"*. A more technical definition would be that automated planning is the discipline of artificial intelligence focused on the development of solutions of mathematical models, being theses ones solved by programs that take the description of a particular instance of a model and automatically compute it's solution.

9.4.3.2.2 Common elements of planners

All the planners have these three components:

- **The conceptual model**, which is the formal definition of the task to be solved as well as the solution's structure.
- **The representation language**, which is the way the problem and the environment are defined.
- **The algorithm**, which is the technique used to solve the problem.

9.4.3.2.3 Computational complexity of planning

In this section, it will be exposed the minimum, maximum and average computational complexity of automated planning.

The minimum computational complexity of automated planning would be a problem with a single operator of size one which it's only effect would be modifying this input predicate (eliminating it or modifying it's value if it's numeric) and an input problem defining a single initial predicate and a single goal predicate. This problem will only have one possible action to do and a single predicate as input choice. All in all, the minimum computational complexity of the most basic input problem possible would be the minimum possible complexity. That is to say, **O(1)**.

As the possibilities of input problems, domains and their corresponding representations is infinite, it is not possible to define an average computational complexity for automated planning.

The maximum computational complexity for automated planning is the maximum possible complexity for any kind of problem, which is **NP-hard**.

Planners used

The planners used in this project are:

- **Metric-FF**: is a domain independent planner that allows to operate with *PDDL 2.1* sentences as well as numeric digits sentences. It is an extension from the planner *FF* used to solve linear tasks. For the search process, it uses the *Enforced Hill Climbing (EHC)* algorithm, which is a variation of *Hill Climbing* algorithm based on local search, which in case of failing in the search of better successors (due to local minimums) it uses breadth first search algorithm.
- **Fast Downward**: its a domain independent progressive planner. It's heuristic it's called casual graph heuristic.
- **LPG**: the planner *Local Search Planner (LPG)* it's a planner whose first version was launched in 2003. The core of this planner is based on stochastic local search and the temporal action graphs (TA_graphs).

9.4.3.2.4 Automatic domain and problems modificators

In this section it will be explained the automatic modificators used for the tests. A more complete overview of the state of the art of these modificators can be found in the Spanish section of the state of the art included in this memory.

The automatic modificators used in the tests are:

- **Automatic macrooperators generators**: they join several operators in a single operator in order to function as a shortcut for the planner. These macro-operators tend to be of a greater size than the original operators, which increases the branching factor. The balance of these beneficts and their losts is called the utility problem. The automatic macrooperator used in the tests is called *Planning Task Transformer (PTT)*.

- **Representation of predicates with numeric digits**: in some of the predicates, it does not matter it's identificator, what it really matters for planning is the number of these predicates. If this is the case, these predicates can be substituted by numerical functions.

- **Representation of predicates with numeric predicates**: this representation is based on the same principle of the numeric representation with digits, the difference is that in this case the numeric representation is not with digits but with predicates. This provides the advantage that while not all planners allow numerical digit problems, all of them can process numerical predicates.

9.4.4 Work developed

9.4.4.1 Technologies used

It has been used software from third parties that in every phase of the development of this work it has always been free software. The programming languages used for the development of this project are:

- **Python**: used for some of the scripts.
- **Bash**: used for some the rest of the scripts.
- **PDDL**: used for the domain and problems representation.

Additionally, LaTeX was used for writing this document, Git was used for the control version of both the code and the memory, Sharelatex was used to edit this document and Gedit was the editor used for programming.

9.4.5 Domains developed

All the domains are modifications of the domain and problems presented to the *IPC* 2014 under the name of *Childsnack*. The summary of all the changes made by all the versions that will be tested is the following one:

- **Original version**: unmodified version of *Childsnack*.
- **Version 01**: it uses macrooperators. It joins the operators `move_tray` with `serve_sandwich` and with `serve_sandwich_no_gluten` , it deletes the operator `move_tray` and includes the operator `move_tray_kitchen` so that trays cannot move between tables.
- **Version 02**: it changes the operator `make_sandwich` so that it cannot accept as input a gluten free component and a gluten free bread in order to avoid labelling sandwich as if they contained gluten when they do not.
- **Version 03**: it uses macrooperators. The only operators that this version has are:
  `make_sandwich_move_tray_serve_sandwich_no_gluten_move_tray_kitchen` , `make_sandwich_move_tray_serve_sandwich_move_tray_kitchen` and `move_tray_kitchen` .
- **Version 05**: manual numeric version with propositions.
- **Version 08**: manual numeric version with digits.

- **Version 09**: it agglutinates several predicates in one. It deletes the predicate `(waiting ?c ?p)` as now the child location it is stored in the predicates `(not_allergic_gluten ?c ?p)` and `(allergic_gluten ?c ?p)` .
- **11 version**: it changes the negative predicates for the negation of the positive ones. The predicates `(not_allergic_gluten ?c)` and `(served ?c)` are now expressed as `(not (allegic_gluten ?c))` and `(not (waiting ?c))` respectively.
- **Version 12**: this version slices the planning process into several phases which cannot be executed until the previous phases are finished.
- **Version PTT**: it is the same version as the original version, as *PTT* was unable to find any macrooperator for this domain.
- **Automated numeric propositions version**: it is generated by *Baggy*.
- **Automated numeric digits version**: it is a numeric digits version generated by a domain independent software whose domain varies slightly from the manual numeric version with digits.

9.4.6 Obtained results

The tests where made with the planners *Metric-FF*, *LPG-td* and *Fast Downward*. However, the results of the execution of the last planner mentioned are not included in this memory as it did solve less than 10% of the total number of all problems from all versions. Therefore, this made it almost impossible to interpret the obtained results.

The metrics used to determine the quality of each version are: the number of problems solved for each planner, the time taken for solving these problems and the quality of the plans. The time limit for each problem was 30 minutes, as it is stipulated in the *IPC* competitions.

The results obtained with these metrics were compared separately among the different planners and these results were easily interpreted in the case of *Metric-FF*, but as *LPG-td* was not deeply understood, the explanations of these results are not so clear.

Finally, in order to prove that the results among these two planners are not correlated, the Pearson correlation index was used to compare the average *IPC* time results of each of the version. The value obtained when performing this operation is **-0.6256**, which means that the versions which are good for one of the planners will outperform with the other one.

9.4.7 Conclusions and future work

In this section, it will be explained the conclusions taken from this project and how this project could be continued.

9.4.7.1 Conclusions

The conclusions obtained from the development of this work could be summarized as follows:

- Variations of the representation of both problems and domains can make a great difference in the planning performance (understanding this performance as the number of problems solved, time taken and quality of these plans).
- There is still a long path to be traveled in the creation of software that automatically changes the representation of both domains and problems, as the results of the automatically modified versions where worse than some of the versions made by hand.
- The suitability of the different representations varies depending on the type of planner used.

9.4.7.2 Future work

The future work to be done in this area would be to make the same type of tests but with more domains and more planners. Then, the long term future work goal would be to program software capable of finding the best representation for each planner.

Bibliografía

[1] N. Rochester y C.E. Shannon J. McCarthy, M. L. Minsky. A proposal for the dartmounth summer research project on artificial intelligence.

[2] Susan Knapp. Artificial intelligence: Past, present, and future.

[3] J. McCarthy. What is artificial intelligence?

[4] Stuart J. Russel y Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[5] A. M. Turing. Computing machinery and intelligence. (English) [On the turing test]. *Oxford University Press*, 59(236):28, 1950.

[6] Home page of the loebner prize in artificial intelligence.

[7] Anotaciones de Ada Augusta L. F. Menabrea. Sketch of the Analytical Engine. (English) [On conjetura de lady lovelace]. *Bibliothèque Universelle de Genève*, (82), 1842.

[8] Mikel Asensio. Los autómatas de hefesto o el procedimiento para crear seres procedimentales. *Íber: Didáctica de las Ciencias Sociales, Geografía e Historia*, 1(2):79–98, 1994.

[9] Sergio Jiménez Cruz. El nuevo paradigma filosófico de la inteligencia artificial. *Revista digital para profesionales de la enseñanza*, (4):20, Septiembre 2009.

[10] Mary Wollstonecraft Shelley. *frankenstein*, volume 149. Ediciones Colihue SRL, 2003.

[11] Isaac Asimov and Manuel Bosch Barrett. *Yo, robot*. Edhasa, 2004.

[12] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[13] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[14] Allen Newell, John C Shaw, and Herbert A Simon. Report on a general problem solving program. In *IFIP Congress*, volume 256, page 64, 1959.

[15] John McCarthy. History of lisp. In *History of programming languages I*, pages 173–185. ACM, 1978.

[16] Edward H Shortliffe, Randall Davis, Stanton G Axline, Bruce G Buchanan, C Cordell Green, and Stanley N Cohen. Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the mycin system. *Computers and biomedical research*, 8(4):303–320, 1975.

[17] Joshua Lederberg. *How DENDRAL was conceived and born*. ACM, 1990.

[18] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[19] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[20] Alain Colmerauer and Philippe Roussel. The birth of prolog. In *History of programming languages—II*, pages 331–367. ACM, 1996.

[21] Mª Jesús Llorens Largo, Faraón y Castel de Haro. *Lógica de primer órden, lógica computacional y ampliación de lógica*. Universidad de Alicante.

[22] Hans Moravec. *Mind children*, volume 375. Cambridge Univ Press, 1988.

[23] Israel Viana. Deep blue, la máquina que ¡¡humilló¿¿ al ser humano. *ABC*, 01/05/2014.

[24] John Markoff. Computer wins on 'jeopardy!': trivial, it's not. *New York Times*, 16, 2011.

[25] Sarah Griffiths. Artificial intelligence breakthrough as google's software beats grandmaster of go, the 'most complex game ever devised. *Daily Mail*, 27/01/2016.

[26] Home page of patrick haslum.

[27] Álvaro Torralba Arias de Reyna. Symbolic search and abstraction heuristics for cost-optimal planning in automated planning, 1 2015.

[28] Moisés Martínez Muñoz. Automated planning through abstractions in real world environments, 2013.

[29] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[30] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[31] Tomás De la Rosa Turbides. Razonamiento basado en casos aplicado a la planificación heurística, 1 2009.

[32] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

[33] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

[34] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[35] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[36] Edwin PD Pednault. Adl and the state-transition model of action. *Journal of logic and computation*, 4(5):467–512, 1994.

[37] Raymond Reiter. A logic for default reasoning. *Artificial intelligence*, 13(1):81–132, 1980.

[38] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins SRI, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language version 1.2. 1998.

[39] Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[40] Metric-ff homepage.

[41] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. 14:253–302, 2001.

[42] Ff and metric-ff in the 3rd international planning competition.

[43] Conformant-ff.

[44] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[45] Alfonso Gerevini e Ivan Serina. Lpg: a planner based on local search for planning graphs. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*, 2002.

[46] Lukáš Chrpa. Generation of macro-operators via investigation of action dependencies in plans. *The Knowledge Engineering Review*, 25(03):281–297, 2010.

[47] Carlos Eduardo Areces, Facundo Bustos, Martín Dominguez, and Jörg Hoffmann. Optimizing planning domains by automatic action schema splitting. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[48] Raquel Fuentetaja and Tomás de la Rosa. Compiling irrelevant objects to counters. special case of creation planning. *AI Communications*, 29(3):435–467, 2016.

[49] Pat Riddle, Jordan Douglas, Mike Barley, and Santiago Franco. Improving performance by reformulating pddl into a bagged representation. *Heuristics and Search for Domain-independent Planning (HSDIP)*, page 28.

[50] Home page of ipc from 2014.

[51] Patricia J Riddle, Robert C Holte, and Michael W Barley. Does representation matter in the planning competition? In *SARA*, 2011.

[52] Pat Riddle, Mike Barley, Santiago Franco, and Jordan Douglas. Bagged representations in pddl. In *4th Workshop on the Intl Planning Competition*, pages 17–23, 2015.

[53] EFE. Las empresas que tengan hasta 250 trabajadores también serán consideradas pymes. *20 Minutos*, 2013.